HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

USTER®
Think quality

USTER Mobile Alerts (Feasibility Study to the Development and Application of Mobile Apps in a Yarn Spinning Mill)

Bachelor Thesis

Computer Science Department
University of Applied Sciences Rapperswil

Spring Term 2015

Author: Philipp Gayko

Advisor: Prof. Hansjörg Huser

External Co-Examiner: Stefan Zettel, Ascentive Zürich

Internal Co-Examiner: Prof. Beat Stettler

Project Partner: Flavio Carraro, Uster Technologies AG

Date: June 12, 2015

Contents

1. Abstract

In all modern yarn spinning mills there is a need for sophisticated production information management and decision support tools. Uster Technologies AG is a global technology company that develops and sells software products targeting Quality management throughout the entire yarn manufacturing process. Software currently under development has the goal of changing the classic principle of the "user going to the PC", to a more modern principle of the "data going to the user".

A convenient and contemporary approach to provide users with relevant information in real-time is to display it on their smartphone or tablet computer. Until now, Uster Technologies AG has not developed any software for mobile devices, nor has the company experiences with the currently available technologies for Mobile Cross-platform Development. The aim of this study is to assess whether it is worthwhile for Uster Technologies AG to develop competencies in Mobile App Development. In doing so, answers to the following questions will be sought: "What is a good way for start?", "What are the risks?" and "What are the decision criterions?"

The feasibility study (UsterMobileAlerts_FeasibilityStudy.docx) answers the above questions and serves an important element of the decision for a potential entry into Mobile App Development at Uster Technologies AG. Furthermore, it provides developers at USTER with useful information on getting started. A suitable technology for Mobile Cross-platform Development at Uster Technologies AG is Xamarin.Forms. The developed USTER Mobile Alerts System is a demonstration and proof of feasibility of this technology. Specially developed simulation software generates Quality Alarms, and an App running on a mobile device receives these alarms and alerts the user. The USTER Mobile Alerts App can run on all common Mobile Platforms including Android, iOS and Windows Phone. The App features a minimal user-friendly UI design that complies with Uster Technologies own Style Guide.

2. Introduction

This feasibility study has the aim to assess whether it is worthwhile for Uster Technologies to develop competencies in Mobile App Development. The created report provides software developers with useful information on getting started with the Mobile Cross-platform Development Software Xamarin. The documentation of this study is divided into two documents (Consider chapter 2.1)

To examine if Xamarin.Forms is a suitable Mobile Cross-platform Development Tool for getting started with Mobile App Development at Uster Technologies AG, an experimental USTER Mobile Alerts App that can run on all common Mobile Platforms including Android, iOS and Windows Phone was developed.

2.1. Use of Documents

To comply with the secrecy directives of Uster Technologies AG and the agreed commitments in chapter **??** the following terms have to be considered:

- The document UsterMobileAlerts_FeasibilityStudy.pdf with all its results and conclusions serves as an element of the decision for the management of Uster Technologies AG. It contains details of software under development that are confidential knowledge of Uster Technologies AG and must not be published.
- The document UsterMobileAlerts_BachelorThesis.pdf contains the administrative parts of this Bachelor Thesis plus a subset of the insights gained with this study. This document can be published and serves as a source of information for software developers they want to start Mobile Cross-platform Development with Xamarin.
- The management summary and further chapters which discuss the business topics of this feasibility study in details can be found in the document UsterMobileAlerts_FeasibilityStudy.pdf.

3. Xamarin

3.1. Introduction

Classic Xamarin and Xamarin.Forms are Mobile Cross-platform Development Tools that enable developers to use .NET C# as the programming language. Source code deployed with Xamarin is shared across iOS, Android and Windows Phone Mobile. Contrary to the approach write apps in native programming languages, the source code of a Portable Library is written once and automatically parsed and compiled against the native platform. Built apps can be installed on mobile phones and tablet computers.

3.2. Xamarin.Forms

The more recent technology Xamarin.Forms has the advantage that the UI source code (XAML is supported) must be written once, whereby the default platform specific UI elements are rendered (Consider chapter 4.2 for screen shots). If it's needed to design a specific look or behavior a custom renderer for the UI elements can be implemented. Using Xamarin.Forms is a very efficient and time-saving way to build simple but stable and good responsive apps with a few limitations to the diversity of UI elements. Xamarin.Forms is relatively new. The first release was in May of 2014. Ongoing development by a dedicated group of Xamarin Engineers ensure the continuing extension of the Xamarin.Forms API.

3.3. Software Development with Xamarin

Xamarin Development requires certain additional hardware and software. Amongst three mobile phones that support the latest versions of iOS, Android and Windows Phone Mobile, a Mac computer as the iOS Build Host is needed. Xamarin licenses for iOS and Android as well as an Apple Developer License have to be purchased before starting Mobile App Development with Xamarin. In order to develop Windows Phone Mobile 8.1 a PC that runs Windows 8.1 has to be used for development.

3.3.1. Installation of the required Tools

It is recommended to use Visual Studio 2015 for Xamarin Development. A preview version of Visual Studio 2015 Ultimate was used for the development of the Experimental USTER Mobile Alerts System. It comes with several Cross-platform Tools and a fast booting Android Emulator that enables to run Android Apps compiled for version 4.4 KitKat or later. In order to develop Windows Phone Mobile 8.1 Apps it is required that Windows Phone SDK 8 is installed (there is an option to check in the Visual Studio Install Wizard when installing on Windows 8.x systems). To getting started with Xamarin Development the following tools have to be installed (keep order!):

1. Visual Studio 2015 on a Windows 8.1 PC or Notebook.
2. Xamarin for Visual Studio.
3. Xcode on the Mac (Register as an Apple Developer and download Xcode).
4. Xamarin for OS X.

Further instructions can be found on the Xamarin webpage.

3.3.2. Development Environment Setup

Android and Windows Phone App Development require only Visual Studio with Xamarin and the required Windows Phone SDK as mentioned in chapter 3.3.1. Development for iOS requires that the PC that runs Visual Studio and the Mac that is used as the iOS Build Host must be connected to the same network. Instructions to connect Visual Studio with the Mac Build Host can be found on the Xamarin webpage. After one of the platform projects was executed in Visual Studio the Emulator Application Window Appears - Android and Windows Phone Emulators on the Windows PC Desktop and iPhone or iPad Emulators on the Mac Desktop. It is recommended that emulators are used to test and debug small changes in the source code. Emulators often behave different from devices and the UI appears different as on devices. Therefore, it is recommended to debug apps directly on the mobile devices for a time-saving and efficient development. In order to debug on Android and Windows Phones devices have to be connected to the Windows PC through the supplied USB data cable. iPhones and iPads have to be connected via the USB data cable to the Mac (Figure 3.1). Consider chapter 3.4.3 for more information about debugging apps on devices.

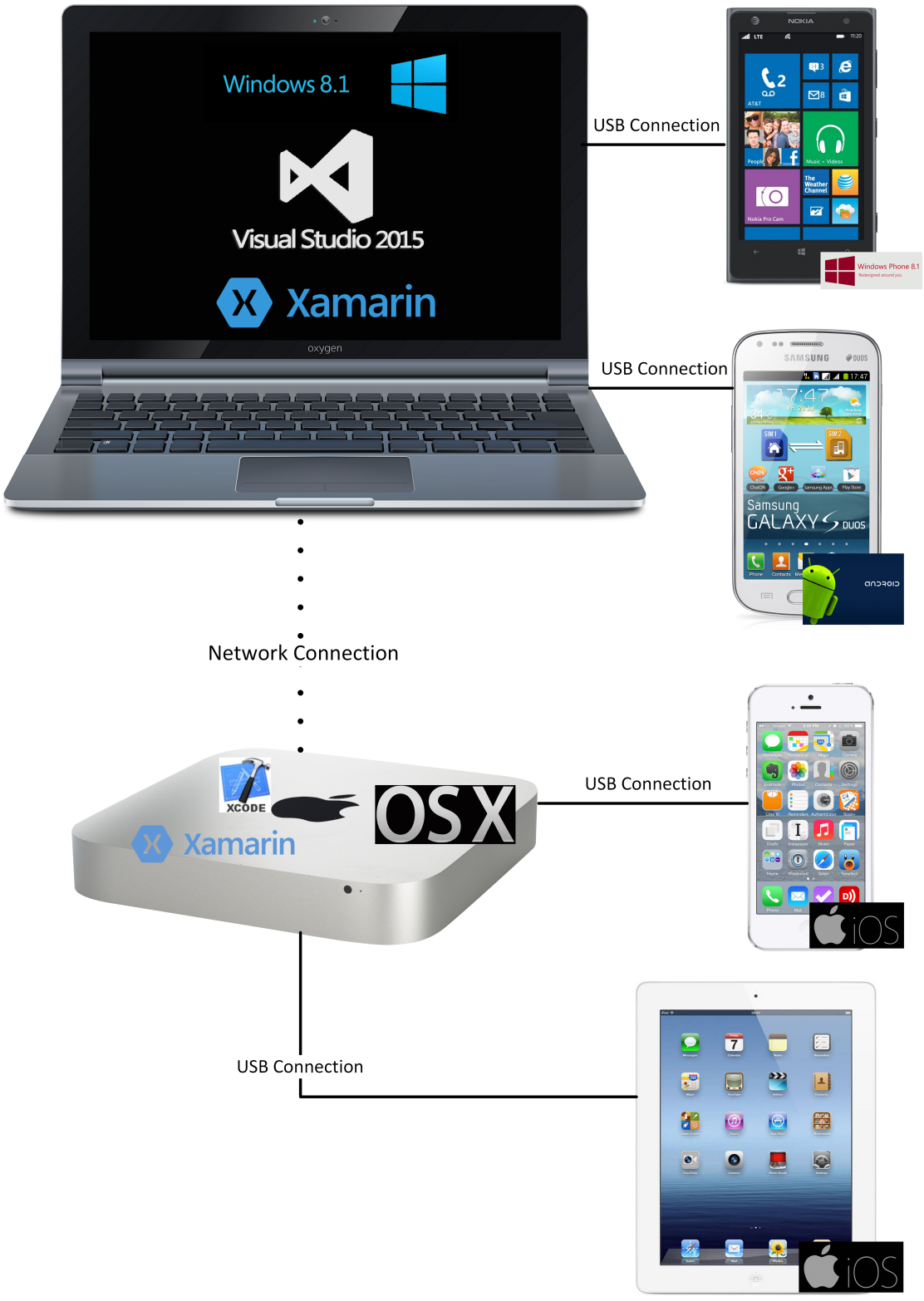


Figure 3.1.: Setup for Development with Xamarin on three Mobile Platforms

3.4. Pitfalls and useful Hints

A typical PC-software developer at USTER has a broad knowledge of the .NET Framework and the Microsoft development tools like Visual Studio. He is an expert in writing source code in C# .NET and uses Frameworks like WCF and WPF frequently. When starting with Mobile App Development with Xamarin he will be probably struggling if he hasn't any experiences. As an extension to the comprehensive documentation with lots of tutorials and sample apps on the Xamarin webpage several some useful hints are provided in the following sub-chapters.

3.4.1. Xamarin Update Channel Settings

To connect Visual Studio with the Xamarin Build Host on the Mac it is necessary that both running the same minor version of Xamarin.iOS. Xamarin Update Channel settings in Visual Studio have to be equal to those in Xamarin Studio on the Mac when updating Xamarin.

3.4.2. Run Apps on Devices

To run apps on devices the following preliminary work for each platform is required:

- Android phones needs an USB driver for debugging (ADB) installed and in the Android menu under "Settings" – "Develop Options" "USB Debugging" has to be enabled.
- Windows Phones have to be registered before apps can be deployed on it.
- iPhones and iPads need a device provisioning before apps can be installed on devices. This is somehow complicated but well explained on the Xamarin webpage provide help.

3.4.3. Debugging on Devices

The following points are good to know, before starting debugging on mobile devices:

- Debugging on devices is a useful feature that works great with Xamarin. If the Portable Library source code has to be debugged then it's good to know that Windows Phone has the fastest app start. It is recommended to debug on Windows Phone whenever possible since all platforms running the same Portable Library source code.
- Normally, the debugger stops by a click on the "stop debugging"-button in Visual Studio. But sometimes the iOS debugger stuck. Therefore, it is suggested, to stop iOS apps that are running in debug mode by the common iOS method which will also stop the process on Visual Studio (double-click on the device home button and wipe the app window up).
- Android debug log output on the Visual Studio console is very useful. It provides developers with information such as resource consumption of the app or warnings are provided like "The main thread is used too many times..."

3.4.4. Xamarin.Forms

Xamarin.Forms is a very efficient way to develop apps for the three different platforms. To avoid mistakes, the following points have to be considered:

- It is necessary that third party libraries are referenced in the Portable Library. If a third party library is referenced in the native platform project assembly, it may happened that version errors occur.
- If a third party library doesn't support the .NETPortable Framework the source code can be compiled against the .NETPortable Framework.

Xamarin.Forms will support the Windows 10 Universal App platform soon. It enables developers that build apps for Windows platforms to share even more code. Currently, Xamarin.Forms Windows 10 UAP is in private beta.

3.4.5. Windows Phone Mobile

Windows Phone Mobile is less well developed compared with iOS and Android therefore the following points might be good to know for developers:

- Windows Phone 8.1 must be developed on a Windows 8.1 PC and requires the installation of Windows Phone 8.1 Development Tools.
- It requires the installation of Windows Phone 8 SDK (Important: It must be installed through the Visual Studio installer!).
- Some functionality that comes for free with iOS and Android has to be implemented in Windows Phone Mobile. Stackable Message Dialogs aren't available for example. It is recommended to test functionality provided by the Xamarin API on Windows Phone Mobile first.

3.4.6. Xamarin on TFS

TFS Continuous Integration (CI) can be used to build a Xamarin Solution but wasn't tested for this feasibility study. A Xamarin tutorial that shows how to create a Continuous Integration Workflow can be found on the Xamarin webpage.

4. USTER Mobile Alerts System

This chapter gives an impression of the developed Experimental USTER Mobile Alerts System. Furthermore, it provides developers with a closer look at the implementation details and the architecture of this Xamarin Cross-platform Solution developed with Visual Studio.

4.1. Introduction

The USTER Mobile Alerts System is a demonstration and proof of feasibility of the Mobile Cross-platform Development Tool Xamarin. Specially developed simulation software generates Quality Alarms, and an app running on a mobile device receives these alarms and alerts the user. The USTER Mobile Alerts App can run on all common Mobile Platforms including Android, iOS and Windows Phone Mobile. The App features a minimal user-friendly UI design that complies with Uster Technologies own Style Guide (Consider Figure 4.1).

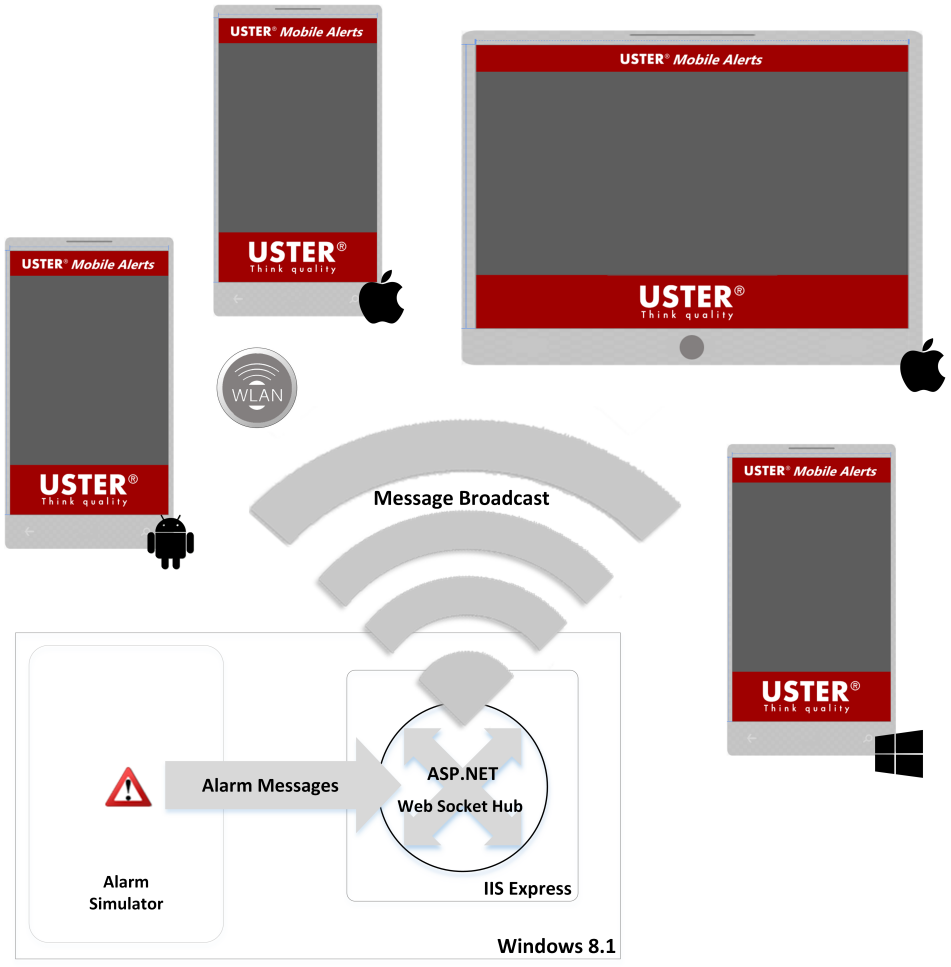


Figure 4.1.: An overview of the Experimental USTER Mobile Alerts System

4.2. Screen Shots

The below screen shots (Figure 4.2) of the App shows the start screen of the Windows Phone App (left outer picture) and the page with the list of received Quality Alarms of the iOS App (picture in the middle). The right outer picture shows the detail page that appears if one of the Quality Alarms was selected.

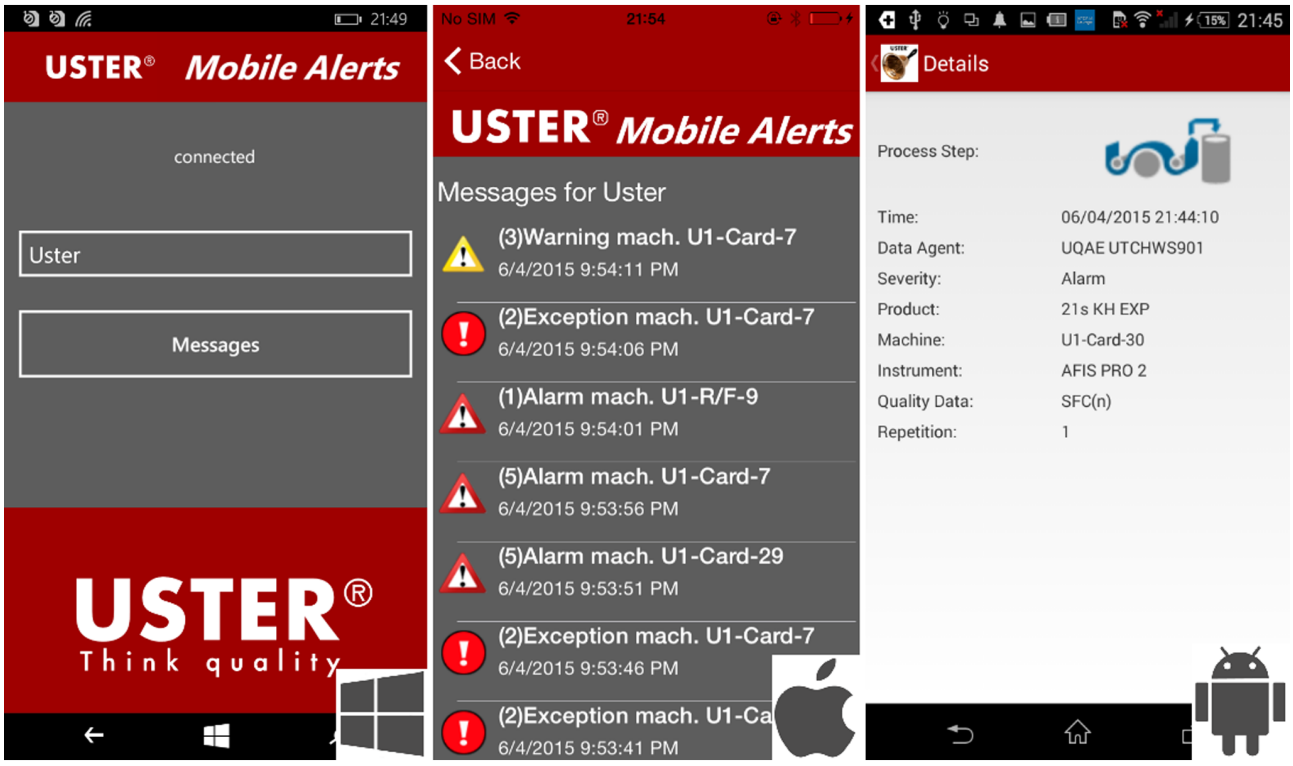


Figure 4.2.: Screen Shots of the USTER Mobile Alerts App of all Platforms

4.3. User Notifications

A great benefit from the use of mobile apps for an alerting system like USTER Mobile Alerts is User Notification. Users are notified about received Quality Alarm messages by ringtones or vibration and are directly linked to the appropriate Quality Alarm when they select the received notification . There are two approaches for User Notifications supported by the API of mobile platforms and Xamarin, Local Notifications and Remote Notifications (Push Notifications).

4.3.1. Local Notifications

Local Notifications are scheduled locally from the app if an event of interest has occurred. If a Quality Alarm was received, the notification method is invoked. Figure 4.3 shows screenshots of the implementation of Local Notification proposals in the experimental USTER Mobile Alerts App. A badge number that displays the number of received Quality Alarms in iOS (1), an Android Task Bar Icon that indicates that Quality Alarms have been received (2) and a notification text element that displays the number of received Alarms (3). This has to be considered as a proposal for User Notification.

Figure 4.3.: Screen Shots of User Notification in the iOS and Android implementation of the USTER Mobile Alerts App

4.3.2. Remote Notifications

Remote Notifications (Push Notifications) are lightweight messages that are sent from a server to the Messaging Cloud Services from platform providers. These services identify devices that are to receive the Push Notification and delivers them to the recipient (Consider Figure 4.4).

Each mobile operating system has its own infrastructure and API's for Remote Notification. Apple uses the Apple Notification Push Service to support Remote Notifications on iOS devices, while Google provides Google Cloud Messaging to enable messaging on Android devices (httt11). Detailed information about this technology can be found on the Xamarin webpage.

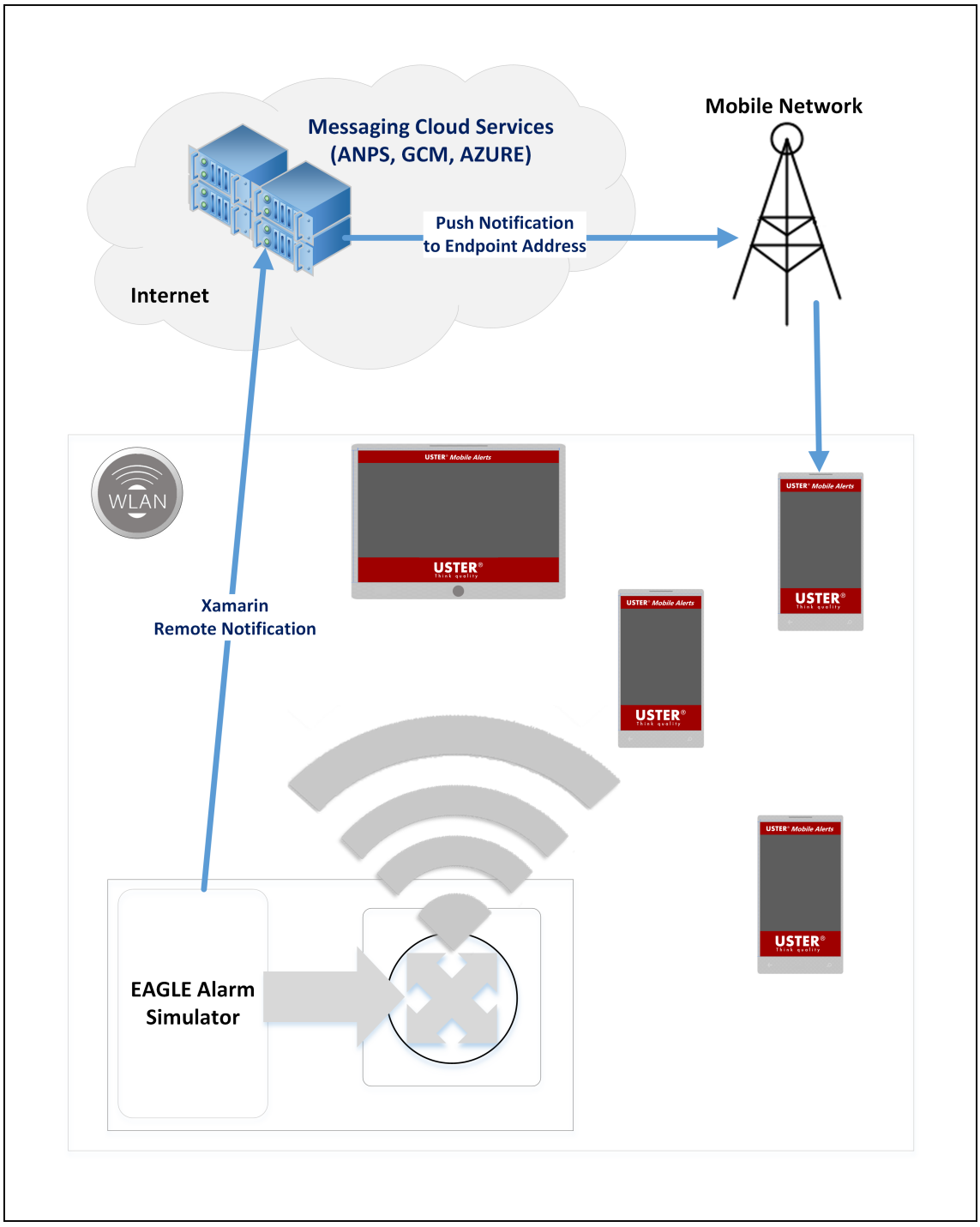


Figure 4.4.: Remote Notification Proposal for USTER Mobile Alerts using Cloud Message Services ANPS, GCM and AZURE

4.4. Communication

The ASP.NET SignalR HTML5 WebSocket library realizes the communication between clients and the Alarm Simulator and enables a stable and reliable bidirectional connection. Clients establish a web connection over Wireless LAN to an ASP.NET SignalR Web Application that is running as a web service on an IIS Web Server and acts as a hub. Using a hub means incoming messages are broadcasted to all connected clients (Consider Figure 4.5). It is conceivable to run the SignalR Hub for USTER Mobile Alerts in another mode than the Broadcast Mode. The Multicast Mode can be used to address only a certain group of users in order to reduce network traffic.

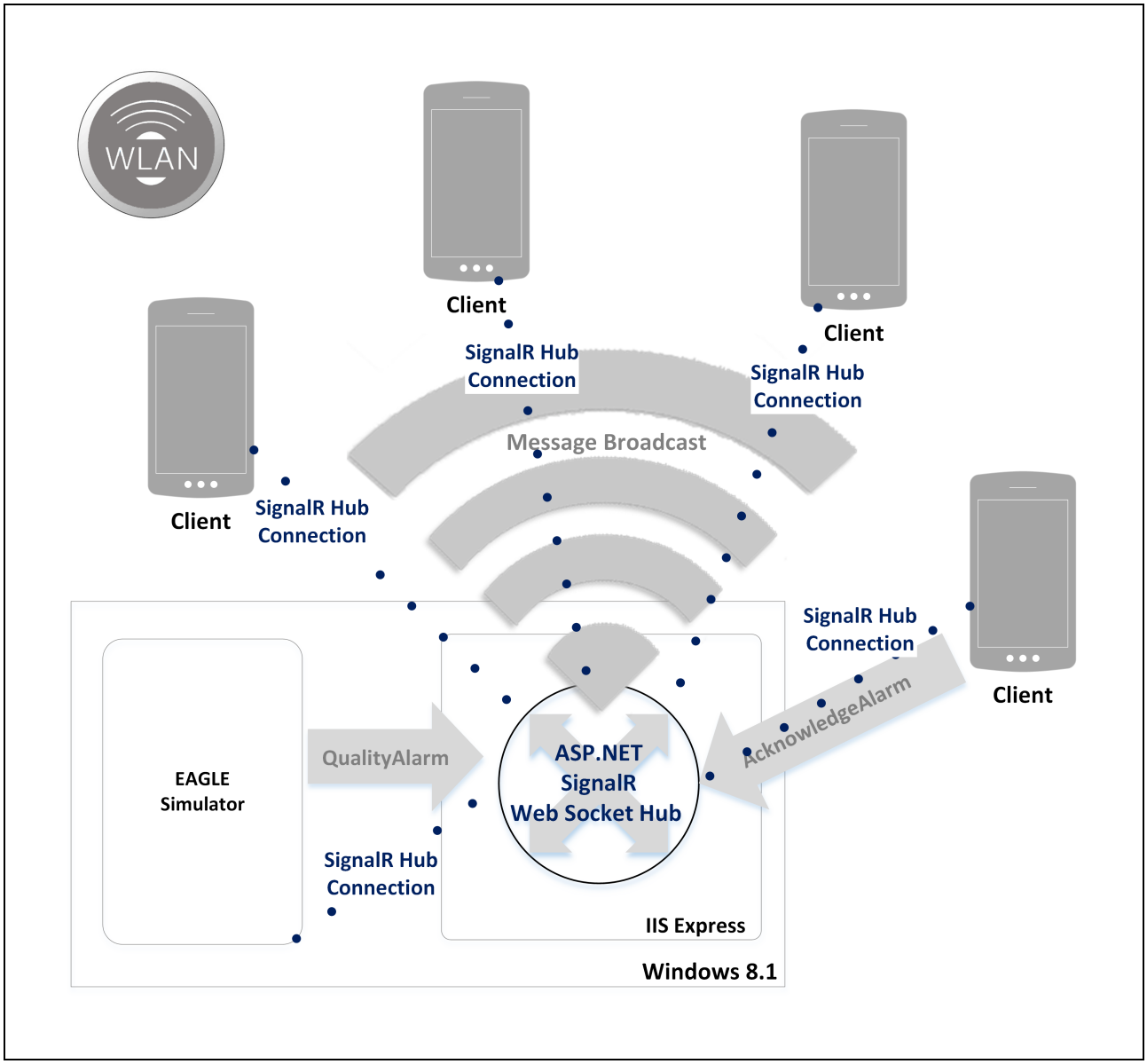


Figure 4.5.: Communication between Clients and the Hub that is running as Web Service

5. Implementation Details of the Experimental USTER Mobile Alerts System

For this feasibility study an experimental USTER Mobile Alerts System was developed. Therefore two approaches were proved. The first approach was Classic Xamarin solution with the development of an iOS and an Android App. A Windows Phone App wasn't developed at this time due to technical problems. The final, more complete system is developed with the Xamarin.Forms technology that enables to build apps for all three platforms with one source code base.

5.1. Visual Studio Solution

The Visual Studio integration of Xamarin allows creating one Visual Studio Solution that contains all needed mobile platform projects (Xamarin.iOS, Xamarin.Android and Windows Phone Mobile). Furthermore, other project types such as ASP.NET Web Applications and the Portable Library project that contains the shared code are also integrated into the same Visual Studio Solution. This enables source control via TFS or Git, Code Completion with the help of the ReSharper Extension, etc.

5.2. First Approach for the USTER Mobile Alerts System developed with Classic Xamarin

For the first approach of the development of the USTER Mobile Alerts App a solution with Classic Xamarin was created. With this technology the domain logic and the model classes are shared through the Portable Library. The code of that library is compiled against the .NET-Portable Framework and referenced in the native platform Classic Xamarin Projects (Consider Figure 5.1). Whereby, the Windows Phone 8.1 (WP 8.1) project isn't a part of Xamarin it references the same Portable Library. The UI source code is a part of the native Xamarin.iOS and Xamarin.Android projects. UI source code is written in C# but uses a wrapping API that adapts the native UI source code. WP 8.1 UI code can be written in XAML or normal C# source code (the WinPhone API differs from the WPF API for desktop applications).

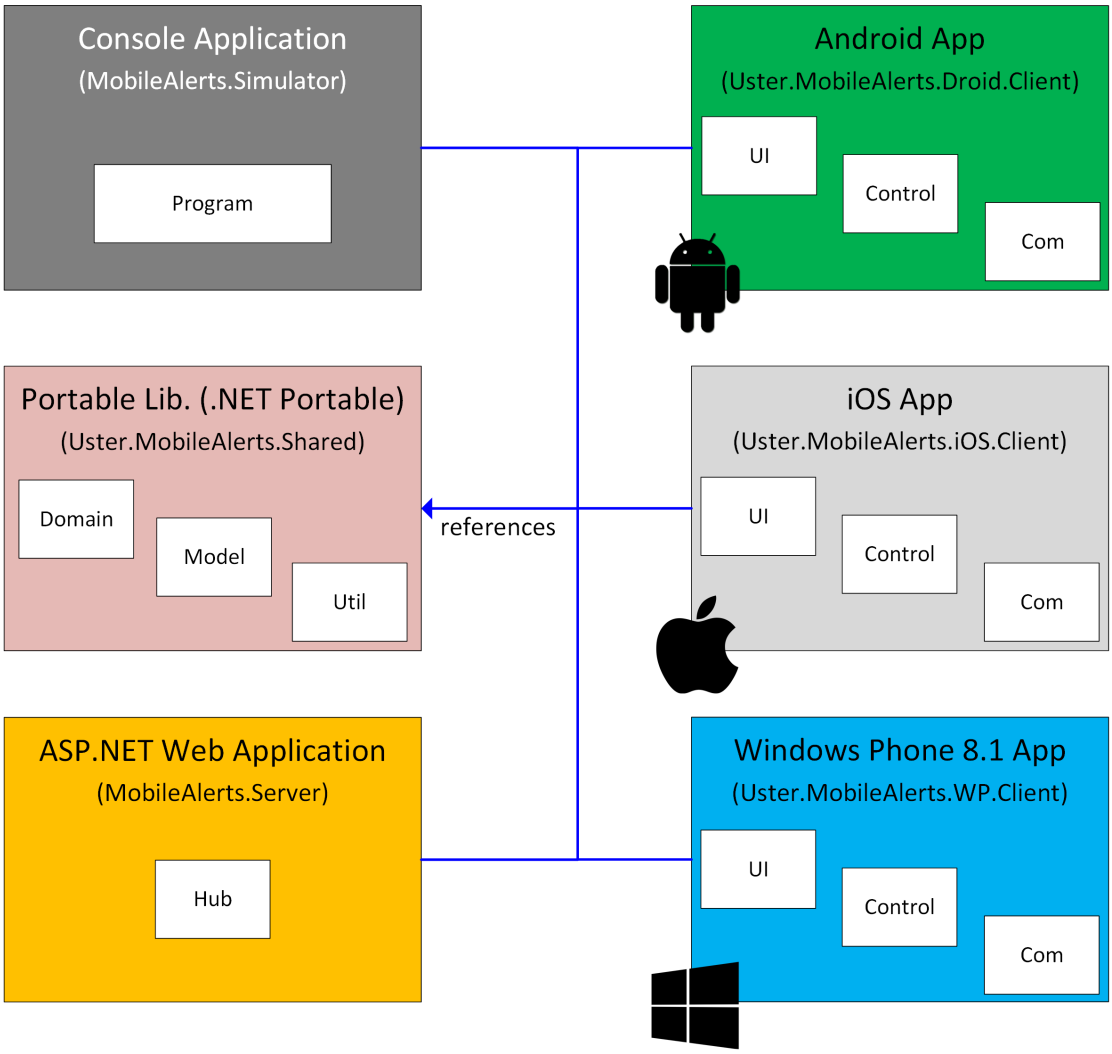


Figure 5.1.: Classic Xamarin Approach. UI and Controller Source Code remain in the native project

5.2.1. Portable Library

The Portable Library Project for this solution contains the code that is shared across the mobile platform projects and also for the Simulator Software (Consider Figure 5.1).

5.2.2. Monotouch.Dialog (iOS)

A simple and easy to understand way to build native iOS UI with Xamarin.iOS is the Monotouch.Dialog Framework. The structure of the UI is organized with nested elements. A RootElement contains a Section Element that can contain another Section Element etc. (Consider Listing 7.1). UI Elements are rendered in their default Look and Feel unless they haven't been modified by Custom Renderers. The below code example taken from the USTER Mobile Alerts Classic Xamarin Solution shows the code for a Quality Alarm element and the resulting UI on a screenshot (Consider Figure 5.2). More information about Monotouch UI Development can be found on the Xamarin webpage.

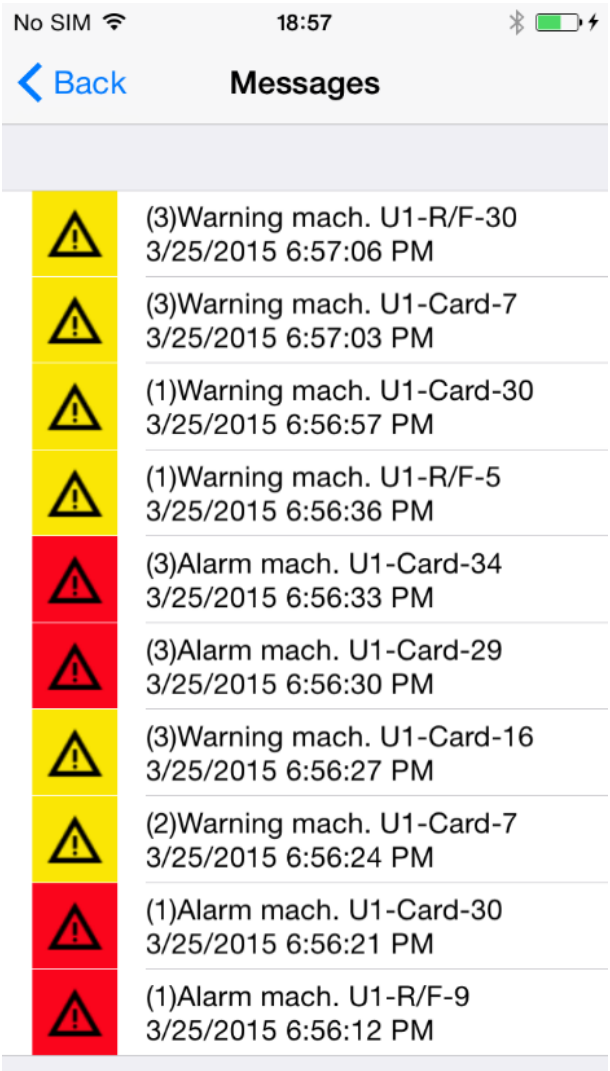


Figure 5.2.: Screenshot of the USTER Mobile Alerts App Start screen implemented with Xamarin.iOS Monotouch.Dialog

```
..
public MessageElementViewController(DialogViewController rootView, MessageDTO messageDto)
{
MessageElement = new StyledMultilineElement(StringUtility.BuildEntryString(messageDto) +
"\n" + messageDto.Timestamp,
() =>
{
var imageView = new UIImageView(new RectangleF(0, 0, 400, 60))
{
ContentMode = UIViewContentMode.Center,
Image = UIImage.FromFile(messageDto.ProcessIconFile).Scale(new CGSize(120, 60)),
};
var np = new DialogViewController(
new RootElement("Details") {
new Section () {
new UIViewElement("Process Step: ", imageView, true),
new StyledMultilineElement(StringUtility.BuildDetailString(messageDto))
{
Font = UIFont.SystemFontOfSize(15f)
}
}
}, true);
rootView.ActivateController(np);
})
{
Image = UIImage.FromFile(messageDto.StatusIconFile),
Font = UIFont.SystemFontOfSize(15f),
SubtitleFont = UIFont.SystemFontOfSize(12f),
};
}
..
```

Listing 5.1: UI code for USTER Mobile Alerts App for one Message Element implemented with Xamarin.iOS Monotouch.Dialog

5.2.3. Xamarin.Android

Xamarin Android uses a declarative approach for UI design. UI Code is organized in Android XML files (.axml) (Listing 7.2 shows the .axml code for the start screen of the App). Visual Studio provides an Android Designer tool similar to the XAML Designer (Figure 5.3 shows the Start Screen UI in the Android Designer). The Controller Code is organized in so called Activities. Each Activity represents one page in the App that belongs to a view (.axml-file). Navigation between pages requires that data is passed by value. Android allows no exchange of references. This approach requires a few more coding and isn't simple to understand. But it is obviously an advantage of Android regarding the Memory Management. Listing 7.3 shows the Handler for Click-events of messages and how is the above mentioned reassignment of data between the Message Page and the Details Page (Figure 17) realized in the source code.

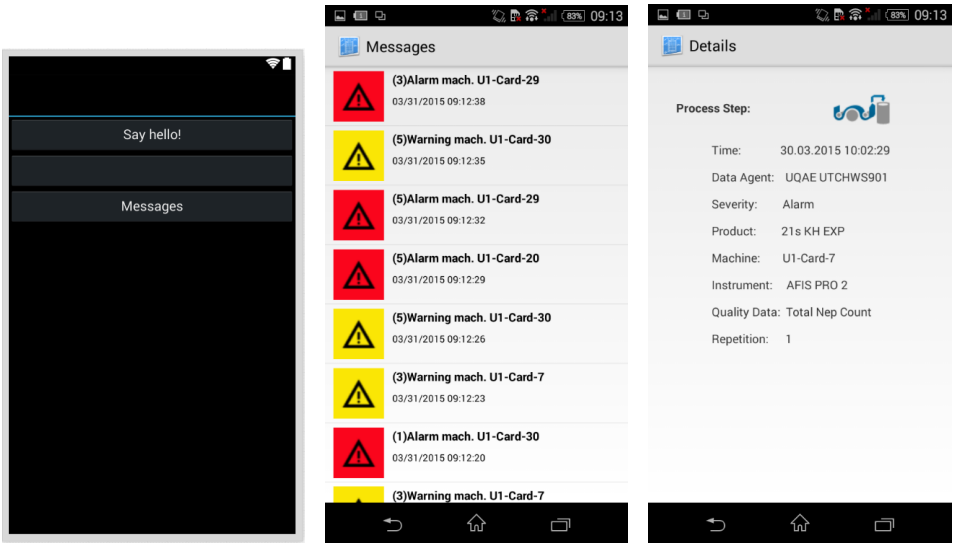


Figure 5.3.: Android Designer, Message Page, Details Page

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:minWidth="25px"
android:minHeight="25px">
<Button
android:text="Say hello!"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/helloButton" />
<Button
android:text=""
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/rolesButton" />
<Button
android:text="Messages"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/messagesButton" />
</LinearLayout>
```

Listing 5.2: Start screen UI code in declarative AXML code

```
..
void OnListItemClick(object sender, AdapterView.ItemClickEventArgs e)
{
var messageDto = MessageBuffer.Messages.ToArray()[e.Position];
var intent = new Intent(this, typeof(DetailsActivity));
intent.PutExtra("Message", StringUtility.BuildDetailString(messageDto));
var imageId = Resources.GetIdentifier(
ProcessIconFilenameConverter.LookupDictionary[messageDto.ProcessIconFile],
"drawable",
PackageName
);
intent.PutExtra("ImageId", imageId);
StartActivity(intent);
}
..
```

Listing 5.3: Code of the Click Handler in the Messages Activity

5.3. Final Approach for the Experimental USTER Mobile Alerts System developed with Xamarin.Forms

The second approach was to test if the first prototype of USTER Mobile Alerts developed with Classic Xamarin can be ported to a Xamarin.Forms Project with the great benefit of getting rid of the burden to modify UI code for each platform separately. Same as Classic Xamarin Xamarin.Forms uses Portable Libraries to share code and also share the UI code (Consider Figure 5.4). Another plus is that the Windows Phone Project is unlike the Classic Xamarin Solution a part of Xamarin and references the same Xamarin.Forms Libraries as Xamarin.iOS and Xamarin.Android which helps to avoid Assembly Errors.

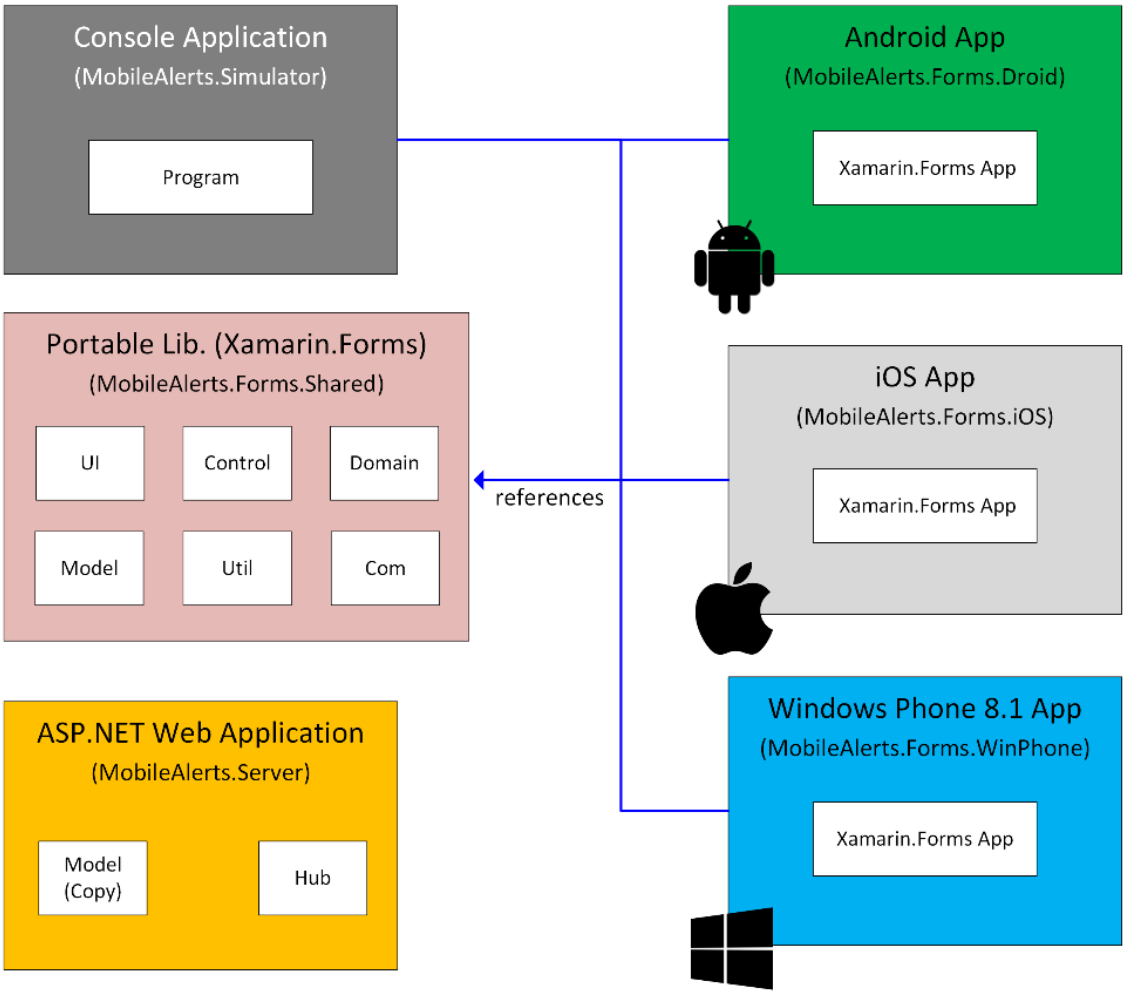


Figure 5.4.: Xamarin.Forms Solution. UI code written in XAML is deployed through the Xamarin.Forms Portable Library

5.3.1. Xamarin.Forms UI

The Xamarin.Forms UI is defined in XAML that is not the same XAML as provided through the WPF Framework. Some classes have different names and properties. The amount of provided UI elements in Xamarin.Forms is limited. But some features like navigation between pages comes for free and have not to be implemented by developers.

5.3.2. Custom Renderers to realize the USTER Logo

The declaration of a Path element in Xamarin.Forms is not supported and requires different implementations on the native platform projects. Whereas, the background color and the red bars can be defined in the XAML code of the Portable Library, Custom Renderers for the USTER Logo and Mobile Alerts Product Logo have to be exported as Path Custom Renderers from the native platform project (Figure 5.5 shows screenshots of the Start Screen with the USTER Logo of the iOS and the Windows Phone app). This is solved through a definition of an empty Class in the Portable Library project that extends a Xamarin.Forms UI Basis Element (Consider Listing 7.4). This Class is visible and can be instantiated in the XAML code (Consider Listing 7.5). The export declaration of the native Path element that contains the name of the empty Class is located before the class declaration as an annotation (Consider Listing 7.6).

Figure 5.5.: Screenshots of the Start Page of the iOS and the Windows Phone App with the USTER Logo

```
..
namespace MobileAlerts.Forms
{
public class UsterLogo : Xamarin.Forms.View
{
}
}
..
```

Listing 5.4: Empty Class in the Portable Library Projects

```
<Grid  Grid.Row="3"
Grid.ColumnSpan="2"
x:Name="BottomBar"
WidthRequest="2000"
HeightRequest="400"
BackgroundColor="#FF9F0000">
<local:UsterLogo/>
</Grid>
```

Listing 5.5: XAML code of the red Bottom Bar UI Element with the imported Customer Renderer “UsterLogo”

```
..
[assembly: ExportRenderer(typeof(UsterLogo), typeof(BottomBarPathRenderer))]

namespace MobileAlerts.Forms.iOS
{
public class BottomBarPathRenderer : VisualElementRenderer<UsterLogo>
{
protected override void OnElementPropertyChanged(object sender, PropertyChangedEventArgs e)
{
..

public override void Draw(CGRect rect)
{
using (var context = UIGraphics.GetCurrentContext())
{
UIColor.White.SetFill();
..
var path = new CGPath();
path.AddLines(new CGPoint[]{
new CGPoint(325.843,98.5032),
..
```

Listing 5.6: Incomplete Custom Renderer Source Code in the iOS Project with the first GCPoint Path Object

Declaration of Path Elements in Android differs from iOS and Windows Phone. Thus, the solution would be to create the USTER Logo and the Product Logo (“Mobile Alerts”) in Adobe Illustrator and export it in the SVG format. The image can then be imported through the svg-android library
(A Xamarin sample can be found here: https://developer.xamarin.com/samples/monodroid/SvgAndro)

5.3.3. Dependency Service for User Notifications

Xamarin.Forms includes a DependencyService to let shared source code to easily resolve Interfaces to platform-specific implementations, allowing to access features of the iOS, Android and Windows Phone SDKs from the Portable Library. This is used for the realization of User Notifications in the USTER Mobile Alerts App. In order to schedule a Local Notification if a Quality Alarm has been received, the Xamarin.Forms App (Portable Library) needs access to the Notification API of the native platforms. This requires an implementation of an interface (Consider Listing 7.7), a registration of the service realized as an implementation of the interface in the native source code (Consider Listing 7.8) and the location of the service in the handler code (Consider Listing 7.9).

```
namespace MobileAlerts.Forms
{
public interface INotify
{
void Notify(int num);
}
}
```

Listing 5.7: DependencyService Interface

```
..
[assembly: Xamarin.Forms.Dependency(typeof(MainActivity))]
namespace MobileAlerts.Forms.Droid
{
..
public void Notify(int num)
{
var context = Xamarin.Forms.Forms.Context as Activity;
if (context == null)
{
return;
}
..
```

Listing 5.8: Implementation of the Interface in the native project code

```
..
public void MessageReceived(object sender, MessageDTO messageDto)
{
//Task.Run(() => { });
if (FilterApplicator.PassQualityMessage(
_filter, messageDto, SettingsViewModel.SelectedRole))
{
MessageViewModel.Insert(messageDto);

Device.BeginInvokeOnMainThread(() =>
{
DependencyService.Get<INotify>().Notify(MessageViewModel.Messages.Count);
});
}
}
..
```

Listing 5.9: Location of the DependencyService and Invocation of the Interface Method

5.3.4. Communication with ASP.NET SignalR

For communication between the Alarm Simuator and the Mobile Clients a reliable and portable technology had to be found. The sample Xamarin app that has been implemented for a first try uses WCF and is able to send messages only in one direction. A sample application that features bidirectional communication via a TCP-Binding and WCF couldn't been found. WCF isn't entirely supported by iOS and Android. A more suitable approach is the use of the HTML5 WebSocket library ASP.NET SignalR. It provides a stable and reliable connection between the SignalR Hub and Clients on all platforms. SignalR also provides a very simple, high-level API for doing server to client RPC (call JavaScript functions in your clients' browsers from server-side .NET code) in your ASP.NET application, as well as adding useful hooks for connection management, e.g. connect/disconnect events, grouping connections, authorization. A detailed description of SignalR can be found here: http://www.asp.net/signalr. Listing 7.10 shows that just some lines of source code are needed to establish a connection between clients and the SignalR Hub that broadcasts all incoming messages.

```
..
[assembly: OwinStartup(typeof(Startup))]

namespace Uster.MobileAlerts.Server.Core
{
public class Startup
{
public void Configuration(IAppBuilder app)
{
app.MapSignalR();
}
}
}


..
namespace Uster.MobileAlerts.Server.Core
{
public class AlertHub : Hub
{
public void Send(MessageDTO messageDto)
{
Clients.All.messageReceived(messageDto);
}
}
}
```

Listing 5.10: This is the only source code that is needed to establish a SignalR Broadcast Hub

The implementation of a client that uses the SignalR HubConnection needs only a few line of source code (Listing 7.11) for a connection with the Hub. The source code of the Connect Method is different for Windows and for Android and iOS. Windows Phone requires the LongPollingTransport Method, because in a few seconds the WP 8.1 Client connects and disconnected several times for any reason.

```
..
namespace MobileAlerts.Forms
{
public class Client
{
private readonly HubConnection _connection;
private readonly IHubProxy _proxy;

public event EventHandler<MessageDTO> OnMessageReceived;

public Client()
{
_connection = new HubConnection("http://192.168.0.192:64491");
_proxy = _connection.CreateHubProxy("AlertHub");
}

public async Task Connect()
{
await _connection.Start();

_proxy.On("messageReceived", (MessageDTO message) =>
{
OnMessageReceived?.Invoke(this, message);
});
}

public async Task ConnectLongPolling()
{
await _connection.Start(new LongPollingTransport());

_proxy.On("messageReceived", (MessageDTO message) =>
{
OnMessageReceived?.Invoke(this, message);
});
}

public Task Send(MessageDTO message)
{
return _proxy.Invoke("Send", message);
}
}
}
```

Listing 5.11: Client source code uses the SignalR HubConnection object

A. Appendix Project

Philipp Gayko | pgayko(at)hsr.ch

A.1. Persönliche Reflexion

Das ich im Rahmen dieser Bachelorarbeit eine wichtige Machbarkeitsstudie im Auftrag meines Arbeitgebers durchführen konnte, hat mich mit Stolz erfüllt. Die vorliegende Studie wird als wichtiges Element für den Entscheid dienen, ob Uster Technologies den Einstieg in die Entwicklung und den Verkauf von Mobile Apps wagen wird. Diese Tatsache hat mich sehr dazu bewogen einen guten Bericht zu erarbeiten, der möglichst alle offenen Fragen beantworten kann.

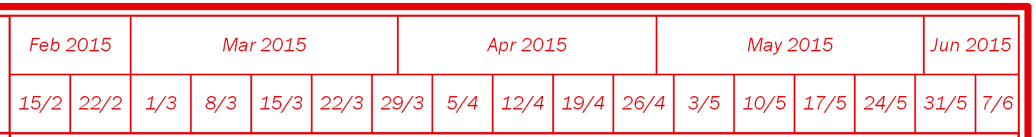
Die Entwicklung des experimentellen USTER Mobile Alerts Systems und einer App mit Hilfe der Cross-Platform Technologie Xamarin hat besondere Freude bereitet, obwohl ich bisher keine Erfahrung auf dem Gebiet von Mobile App Entwicklung hatte. Anfänglich hatte ich deshalb auch Schwierigkeiten die richtigen Ansätze zu wählen. Einerseits habe ich mich zuerst zu lange mit der Suche nach einer Technologie, welche für die Kommunikation zwischen den Clients verantwortlich sein soll, aufgehalten. Andererseits lies ich mich durch eine irreführende Beispiel-Applikation davon abhalten, die sehr gut geeignete Xamarin.Forms Technologie von Anfang an für USTER Mobile Alerts zu verwenden. Schlussendlich muss ich aber sagen, dass man aus Fehlern lernt und diese Erfahrungen in den Bericht eingeflossen sind. Sie werden anderen Xamarin-Entwicklern helfen, einen guten Einstieg in diese Technologie zu finden.

Nicht zuletzt möchte ich auch Professor Huser und Flavio Carraro für die gute Zusammenarbeit danken. Die wöchentlichen Sitzungen haben mir einen guten Projektrahmen gegeben und die nützlichen Inputs und Verbesserungsvorschläge sind Teil dieser Arbeit.

Zum Abschluss des Projekts kann ich sagen, dass ein sehr wichtiges Ziel bestimmt erreicht wurde, es ist bereits jetzt eine rege Diskussion in der Uster Technologies AG rund um dieses Thema in Gange.

A.2. Project Schedule

ID	Phase Name	Start	Finish
1	M0: Start	17.02.2015	17.02.2015
2	Setup Project and Schedule	16.02.2015	24.02.2015
3	M1: Review Project Schedule (time line)	24.02.2015	24.02.2015
4	Xamarin familiarization	19.02.2015	27.02.2015
5	M2: Review Documentation structure, Expectation check	10.03.2015	10.03.2015
6	Define Management Summary	25.02.2015	06.04.2015
7	M3: Review Management Summary	07.04.2015	07.04.2015
8	Create and improve App	25.02.2015	04.06.2015
9	M4: Review runnable Prototype	28.04.2015	28.04.2015
10	Deployment Recherche: App Stores	28.04.2015	03.06.2015
11	Review and document Feasibility	07.04.2015	08.06.2015
12	M5: Review Feasibility Study	09.06.2015	09.06.2015
13	Documentation	02.03.2015	11.06.2015
14	M6: End of Project	12.06.2015	12.06.2015



A.3. Milestones

M1: Project Schedule

Reviewed artefact:

- Schedule (time line)

M2: Documentation Structure

Reviewed artefacts:

- Title page
- Content structure
- Glossary format
- List of sources format
- Agreement (review 17.03)
- Feasibility study content structure (review 17.03)
- Milestones (review 17.03)

Acceptance criteria:

- Title page: Contains all relevant information
- Content structure: Is according student research project HS2014
- Glossary: In a suitable format (any constraints?)
- List of sources: In a suitable format (any constraints?)
- Agreement: All parties agree ready for signing
- Feasibility study: Satisfying the demands of Uster Technologies AG
- Milestones: Are measurable and acceptance criteria are defined

M3: Management Summary

Acceptance criteria:

- The relevant business questions are raised on CEO level
- The decisive criterions are defined
- The technical topics are clear

M4: Runnable Prototype

Reviewed artefacts:

- Simulator that emits mocked Alarms
- Running Apps on iOS and Android mobile devices

	Acceptance criteria: • A simulation software sends mocked Alarms • A server software emits the simulated Alarms over WLAN • App prototypes on iOS and Android receive and display Alarms
M5: Feasibility Study	Acceptance criteria: • One document that contains all relevant information for Uster Technologies is created • One document that contains the administrative part and a subset of the information of the Uster Technologies document is created

A.4. Time Exposure

Total hours of work for this Bachelor Thesis: 410 (360 hours is the effort that is expected)

Listing of sources and literature

[1] UsterMobileAlerts_FeasibilityStudy.docx

[2] http://usternet/Pages/Home.aspx

[3] http://www.developereconomics.com/pros-cons-top-5-cross-platform-tools/.

[4] http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

[5] http://en.wikipedia.org/wiki/.

[6] http://xamarin.com/forms.

[7] http://developer.xamarin.com/.

[8] http://www.s3-us-west-2.amazonaws.com,4.bp.blogspot.com,cdn2.expertreviews.co.uk

[9] www.thephonetown.com,asphostportal.com,www.iwaredesigns.co.uk

[10] www.ayeshacurry.com,www2.pcmag.com,cdn1.knowyourmobile.com,ak1.ostkcdn.com.

[11] https://msdn.microsoft.com/en-us/library/gg597391

[12] https://msdn.microsoft.com/en-us/library/ms731082(v=vs.110).aspx.

[13] http://signalr.net/.

[14] http://sharpmobilecode.com.

[15] http://blogs.edwardwilde.com.

[16] http://www.wintellect.com.

[17] http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

[18] http://mobiledan.net/2012/03/02/5-options-for-distributing-ios-apps-to-a-limited-audience-legally/.

[19] http://venturebeat.com/2014/11/20/the-windows-10-store-will-let-businesses-buy-apps-in-bulk-and-offer-private-sections-for-their-own-apps/.

[20] https://msdn.microsoft.com/en-us/library/windows/apps/jj206943(v=vs.105).aspx.

Glossary

ASP.NET Is an open source server-side Web application framework designed for Web development to produce dynamic Web pages. 14

CI Is the practice, in software engineering, of merging all developer working copies with a shared mainline several times a day. 9

HTML5 Is a core technology markup language of the Internet used for structuring and presenting content for the World Wide Web. 14

Style Guide Red bars with the USTER logo and the product name. 10

UI User Interface. The graphical surface of an application. 10, 16

Uster Uster Technologies AG. World market leader in measure and control the quality of fibers and yarns in the textile industry. 1, 4, 10

Xamarin.Forms Is a Mobile Cross-platforms Development Tool that allows to build mobile apps for Android, iOS and Windows Phone Mobile. Source code (even the UI source code) can be written in C# .NET and shared across the three platforms (http://xamarin.com/forms). 1, 4

XAML Extensible Application Markup Language. Is a declarative XML-based language developed by Microsoft that is used for initializing structured values and objects. 16