Entwickeln für Windows Phone 7

Am Beispiel einer OpenStreetMap Anwendung

Martin Rauscher

23. Mai 2011

1 Einführung

Mit Windows Phone 7 (WP7) hat Microsoft sein Betriebssystem für Mobilgeräte komplett überarbeitet, um der Herausforderung durch Android und iOS gewachsen zu sein. Mit diesem neuen Betriebssystem wird ein neues Entwicklungsmodell eingeführt, welches hauptsächlich auf Microsofts Silverlight-Technologie basiert.

Im Rahmen dieser Arbeit soll WP7 - und die Entwicklung dafür - vorgestellt werden.

1.1 Vorstellung der Beispielanwendung

Um die Konzepte der Entwicklung für WP7 zu testen, wurde eine Anwendung erstellt, die das Betrachten von Karten von OpenStreetMap (und anderen Anbietern) ermöglicht. Sie erlaubt die Steuerung durch die üblichen Multi-Touch-Gesten, das Suchen nach Orten und das Verwalten von Favoriten. Des Weiteren können Routen geplant werden.

Alle Beispiele in dieser Arbeit sind aus dem Quelltext dieser Anwendung. Für nähere Informationen zur “OpenStreetApp” wird auf [1] verwiesen.

1.2 Rahmen dieser Arbeit

Es gibt zwei Arten von WP7 Apps[1]: Zum einen Apps auf Basis von Silverlight und zum anderen Apps auf Basis des XNA Frameworks - einer gekapselte DirectX-Schnittstelle, welche hauptsächlich für die Entwicklung von Spielen verwendet wird.

[1] Für spezielle Microsoft Partner ist es auch noch möglich halb-native Anwendungen zu schreiben.

Diese Arbeit beschäftigt sich ausschließlich mit der Entwicklung von Silverlight Apps, da es nur in seltenen Fällen Sinn macht, mit XNA Nicht-Spiele zu entwickeln und das Entwicklungsmodell von XNA gänzlich anders ist.

2 Windows Phone 7

Windows Phone 7 ist hinsichtlich der Benutzeroberfläche als auch der Anwendungsplattform eine vollständige Neuentwicklung. Der Kern des Betriebssystems hingegen ist eine veränderte Version von Windows EC 7.0, dem Nachfolger von Windows CE 6, worauf Windows Mobile 6.5 beruht.[2]

Gründe nicht auf das bestehende Windows Mobile aufzusetzen waren:

- UI-Konzept zu stark desktop-orientiert
- Fehlende Multi-Touch Unterstützung
- Aufwendig, modernes Look-and-Feel zu erreichen

WP7 begegnet diesen Problemen mit einer starken Multi-Touch-Unterstützung in Silverlight und vordefinierten Kontrollelementen, welche es vereinfachen sollen moderne Anwendungen zu schreiben, die sich in das UI-Konzept einfügen.

3 Silverlight und die Windows Presentation Foundation

Das .NET Framework auf WP7 Handys basiert auf dem - vom Desktop bekannten - Silverlight, welches wiederum auf der Windows Presentation Foundation basiert. In den folgen Abschnitten soll kurz aufgezeigt werden, wie die beiden Frameworks entstanden sind und worin die groben Unterschiede bestehen.[3]

3.1 Windows Presentation Foundation (WPF)

Ende 2006 stellte Microsoft die finale Version 3.0 des .NET Framework vor. Mit diesem wurde die, bis dahin unter dem Codenamen Avalon entwickelte, Windows Presentation Foundation (WPF) der Öffentlichkeit zugänglich gemacht. Microsoft hoffte, dass Entwickler, insbesondere für das nur zwei Monate später erscheinende Windows Vista, nun hauptsächlich mit .NET und WPF entwickeln würden. Bis heute ist allerdings die Akzeptanz von WPF im Consumerbereich eher gering.

Mit WPF führte Microsoft erstmals ein deklaratives Modell zur Gestaltung von Benutzeroberflächen ein. Dabei wird die GUI und die Datenbindungen mit einem auf XML basierenden Format beschrieben. Dies erhöht die Wartbarkeit und Wiederverwendbarkeit verglichen mit den bisher verwendeten, zum Teil riesigen, automatisch generierten Funktionen enorm.

Eine weitere große Neuerung ist die Abkehr von GDI für das GUI-Rendering. Mit WPF wird erstmals vollständig auf (hardwarebeschleunigtes) DirectX gesetzt. Das hat den Vorteil, dass komplexe Oberflächen gerendert, transformiert und mit Effekten versehen werden können, ohne dass die CPU zusätzlich belastet wird.

3.2 Silverlight (WPF/E)

2007 stellte Microsoft Silverlight vor, welches plakativ gesprochen, eine stark reduzierte Version des .NET Frameworks und eine Untermenge[2] der WPF darstellt.

Auch wenn Silverlight oft als Konkurrenz zu Adobes Flash Player gesehen wird, wurde es hauptsächlich mit dem Ziel entwickelt, sogenannte Rich Internet Applications (RIAs) zu ermöglichen, welche hauptsächlich im Geschäftsbereich eine Rolle spielen. Der einzige Bereich in dem Silverlight momentan tatsächlich in direkter Konkurrenz zu Flash steht, ist die Videowiedergabe. Sowohl Smooth Streaming[3] als auch hardwarebeschleunigte Videowiedergabe waren zuerst in Silverlight möglich. Deshalb hat Microsoft einige prominente Unterstützter gefunden: Unter anderem wurden die Olympischen Winterspiele 2008 und einige amerikanische Sportgroßereignisse exklusiv via Silverlight gestreamt; des weiteren setzt auch das deutsche Videoportal "Maxdome" auf Silverlight.

Silverlight wurde von Grund auf plattformunabhängig konzipiert. Microsoft stellt es für die meisten Windows Versionen sowie für Mac OS X zur Verfügung. Des Weiteren sind ein Großteil der Spezifikationen und einige Teile des Codes öffentlich zugänglich. Darauf aufbauend entwickelt das Mono Projekt unter Leitung von Novell eine Linux Portierung. "Moonlight" ist momentan kompatibel mit Silverlight 3.[4]

[2] Aufgrund von unterschiedlichen Releasezyklen fehlen der aktuellen WPF aber einige Klassen, die bereits in Silverlight enthalten sind. Diese wurden aber über ein Silverlight Toolkit nachgereicht.

[3] Bei Smooth Streaming handelt es sich um eine Technik die Videomaterial dynamisch neu kodiert, um sich der zur Verfügung stehenden Bandbreite anzupassen.

3.3 Unterschiede

Trotz der Ankündigung von Microsoft, die Unterschiede zwischen WPF und Silverlight im Laufe der Zeit zu verringern, sind diese momentan noch stark sichtbar, da auf WP7 nur SL3 bzw. ab Herbst SL4 verfügbar ist[4].

Ein beliebtes Feature in WPF, zur Trennung von GUIs vom Verhalten, sind so genannte *Commands*. Diese können an viele Elemente gebunden werden um bei Aktivierung des Elements eine bestimmte Aktion auszulösen und um zu signalisieren, ob diese Aktion im Moment möglich ist. Dieses Feature ist ab SL4 verfügbar.

Ein ebenfalls erst ab SL4 verfügbares Feature, sind *Markup Extensions*. Auf dieses Thema wird noch genauer in Kapitel 4.2.3 eingegangen.

Die beiden größten Unterschied in der GUI-Entwicklung sind zum einen das Fehlen von Triggern, die z.B. Animation bei bestimmten Ereignissen auslösen können, und zum anderen, dass Styles grundsätzlich explizit gesetzt werden müssen und nicht automatisch aus den gegebenen Typen ermittelt werden können.

Diese Auflistung enthält nur die wichtigsten Unterschiede. Für eine detaillierte und vollständige Liste wird auf [5] verwiesen.

4 Entwickeln mit SILVERLIGHT

4.1 Struktur einer SILVERLIGHT Anwendung

Eine SILVERLIGHT Anwendung wird immer in Form einer ZIP-Datei mit der Endung .XAP verteilt. In dieser Datei sind die vier Bestandteile einer App untergebracht:

- Das Manifest (WMAppManifest.xml), welches die Fähigkeiten der App festlegt
- Das Anwendungsobjekt (App.cs und App.xaml), welches die zentrale Steuereinheit bildet
- Die verschiedenen “Seiten” der App als <PageName>.xaml Dateien
- Die Ressourcen (Bilder, Töne, etc.)

Beim Start der App durch das Betriebssystem lädt dieses das Manifest und registriert die darin vermerkten Fähigkeiten. Selbige beschränken z.B. den Zugriff auf den GPS-Sensor oder die Bilder des Nutzers und werden ihm vor der Installation der App angezeigt.

[4] SL5 wird viele der aufgezählten Unterschiede ausgleichen.

Danach wird das Programm initialisiert und die Startseite der App geladen, welche ebenfalls im Manifest vermerkt ist.

Ab diesem Zeitpunkt kann der Entwickler mit dem Navigationsservice zu anderen Seiten navigieren. Die Navigation sollte dabei immer linear sein, damit der Benutzer beim Drücken des - in jedem WP7 Gerät verbauten - Zurück-Knopfs keine unlogischen Sprünge verursacht.

4.2 XAML

XAML ist eine deklarative Sprache zur Beschreibung von Datenstrukturen, die insbesondere für SILVERLIGHT- und WPF-Benutzeroberflächen verwendet wird. Sie ist ein XML Dialekt, der um verschiedene Konzepte erweitert wurde:

4.2.1 Komplexe Attributdefinitionen

In XML können Attributen nur Strings, Zahlen und Referenzen zugewiesen werden, während der Inhalt von Element beliebig komplex sein kann. In XAML können Attribute durch eine simple Erweiterung der Semantik von Elementnamen ebenfalls mit beliebigem Code beschrieben werden. Im Beispiel 1 wird so auf das Attribut `Background` über `Grid.Background` zugegriffen.

Algorithmus 1 Beispiel für Komplexe Attributdefinitionen

```
<Grid Background="Transparent"/>
<!-- oder -->
<Grid>
    <Grid.Background>
        <SolidColorBrush Color="Transparent"/>
    </Grid.Background>
</Grid>
```

4.2.2 Attached Properties

Insbesondere bei der Beschreibung von Benutzeroberflächen steht man oft vor dem Problem, dass Elemente eine Eigenschaft besitzen, die nur innerhalb eines bestimmten Kontextes sinnvoll sind.

Z.B. hat ein Button der in einem *Canvas* angeordnet ist eine X und eine Y Koordinate. Wird er aber außerhalb eines *Canvas* eingesetzt, sind diese Eigenschaften u.U. nicht sinnvoll.

Algorithmus 2 Beispiel für ein Attached Property

```
<Canvas>
    <Button Canvas.Top="100">
        <Canvas.Left>25</Canvas.Left>
        <TextBlock Text="Ein Knopf"/>
    </Button>
</Canvas>
```

XAML bietet für dieses Problem eine einfache Lösung: Genau wie bei komplexen Attributdefinitionen wird hier der Name des Attributbesitzers verwendet um die Herkunft des Attributs zu qualifizieren. Der Wert des Attributs wird dabei in einer statischen Eigenschaft der entsprechenden Klasse gespeichert. [6]

Die sogenannten *Attached Properties* sind eine Spezialform von *Dependency Properties* (siehe 4.3.1).

4.2.3 Markup Extensions

Markup Extensions sind ein Konzept, das es ermöglicht den XAML Parser in begrenztem Umfang zu erweitern, so dass die Verwendung von Code oder komplexen Attributdefinitionen entfallen kann.

Algorithmus 3 Beispiel für eine Markup Extension

```
<TextBlock Text="{StaticResource AppName}" />
<!-- oder -->
<TextBlock Text="{StaticResource ResourceKey=AppName}" />
<!-- vs. -->
this.textblock.Text = (string)this.Resources["AppName"];
```

Markup Extensions werden durch führende und endende geschweifte Klammern gekennzeichnet. Das erste Wort in den Klammern gibt die Klasse der zu verwendenden Extension an. Parameter werden nach dem Klassennamen gelistet und werden in der Form “Parameter=Wert” angegeben. Einzige Ausnahme ist der Standardparameter, der ohne Angabe seines Namens verwendet werden kann. Im Beispiel 3 ist der Standardparameter “ResourceKey”.

Bei jeder Verwendung wird eine neue Instanz der entsprechenden Klasse erzeugt, die Parameter als Eigenschaften gesetzt und eine Methode aufgerufen, die mit beliebigem Code einen Wert für das zu belegende Attribut erzeugt.

SILVERLIGHT bietet in der aktuellen Version keine Möglichkeit eigene *Markup Extensions* zu definieren[5]. Es stehen nur die folgenden Vordefinierten zur Verfügung: *Binding*, *StaticResource* (und *DynamicResource*). Auf deren Bedeutung wird im folgenden Abschnitt eingegangen; außerdem wird aufgezeigt, dass sie beliebig geschachtelt werden können.

4.3 Datenbindung

Eine der größten Stärken von SILVERLIGHT ist die Möglichkeit, Eigenschaften von Elementen auf einfache Weise an Attribute von Datenobjekten oder an Eigenschaften anderer GUI Elemente zu binden. Somit muss kein Code geschrieben werden, um die Benutzeroberfläche bei Änderungen an den Daten aktuell zu halten.

Algorithmus 4 Beispiel für eine Datenbindung

```
<TextBox Text="{Binding Path=Vorname,
                        Mode=TwoWay,
                        Converter={StaticResource conv1}}"/>
```

In Beispiel 4 kann man sehen, wie die Definition einer Datenbindung mit Hilfe der {*Binding*} Markup Extension aussieht. Dabei bezieht sich die Datenbindung auf den aktuellen Datenkontext, da keine andere Quelle angegeben wurde. Dieser Kontext ergibt sich aus dem Element innerhalb dessen das Binding definiert wurde. Jedes GUI-Element[6] hat eine DATACONTEXT Eigenschaft, die es an seine Kind-Elemente weitervererbt, von ihnen aber auch überschrieben werden kann.

Der `Mode` Parameter im Beispiel beschreibt dabei die Richtung der Bindung. Es können beide Seiten den Wert ändern (`TwoWay`) oder nur die Quelle (`OneWay`) oder auch das Ziel (`OneWayToSource`).

4.3.1 Dependency Properties

Ein Problem vieler Datenbindungs-Ansätze ist, dass sie Reflection[7] verwenden, da die Eigenschaften als String angegeben werden. Außerdem muss jede Klasse typischerweise ein Interface implementieren, das die eventuellen Ziele einer Datenbindung benachrichtigt.

[5] Auch diese Funktion ist für SL5 angekündigt.

[6] Genauer, jede von FRAMEWORKELEMENT erbende Klasse.

[7] Reflection bezeichnet die Möglichkeit Objekte einer statisch getypten Sprache zur Laufzeit dynamisch zu untersuchen und zu verändern. Hierfür sind meistens aufwendige Aufrufe an die zu Grunde liegende Plattform zu richten.

In SILVERLIGHT wird dieses Problem mit so genannten Dependency Properties (DPs) gelöst. Im Grunde sind DPs Dictionaries, die als statische Eigenschaften in der Klasse definiert werden. Dieses weißt dann einer Instanz einen bestimmten Wert zu.

Der große Vorteil von DPs ist, dass sich beliebige Objekte registrieren können, um benachrichtigt zu werden, wenn sich der Wert einer Eigenschaft eines bestimmten Objekts ändert; und zwar mit statisch getypten Vorher-Nachher-Werten.

Außerdem lassen sich bei der Definition von DPs Standardwerte festlegen und automatisches Vererben von Werten auf Kind-Objekte in der UI-Hierarchie aktivieren.

4.3.2 Daten-Konverter

Es ist oft wünschenswert zwei Eigenschaften miteinander zu verbinden, die nicht den gleichen Datentyp haben. Das typische Beispiel hierfür ist die Text Eigenschaft eines Textblocks und ein Zahlwert eines Objekts. Im Gegensatz zu diesem einfachen Beispiel, bei welchem die Konvertierung natürlich automatisch passiert, ist es oft komplizierter.

Algorithmus 5 Beispiel für einen Converter

```
public class VisibilityConverter : IValueConverter
{
   public Object Convert(Object value,
                  Type targetType,
                  Object parameter,
                  System.Globalization.CultureInfo culture)
   {
      return (bool)value == true ?
            System.Windows.Visibility.Visible
            : System.Windows.Visibility.Collapsed;
   }

   public Object ConvertBack(Object value,
                     Type targetType,
                     Object parameter,
                     System.Globalization.CultureInfo culture)
   {
      return (System.Windows.Visibility)value
         == System.Windows.Visibility.Visible;
   }
}
```

Dieses Problem wird mit so genannte Konvertern, die beliebige Typen in einander umwandeln können, gelöst. Konverter sind Klassen, welche das *IValueConverter* Interface implementieren. Im Beispiel 5 kann man exemplarisch sehen, wie ein Wahrheitswert in einen Aufzählungstyp - und umgekehrt - umgewandelt werden kann. Mit Hilfe der *Convert* und *ConvertBack* Methoden kann beliebig komplexer Code dazu verwendet werden Werte umzuwandeln.

In Beispiel 4 wurde bereits gezeigt, wie Konverter verwendet werden. Üblicherweise müssen selbige dafür als eine Ressource definiert werden.

4.3.3 Daten-Templates

Will man mehrere Objekte, z.B. in einer Liste, anzeigen, so reichen die Fähigkeiten von Konvertern normalerweise nicht aus. Für diese Anwendung bietet SILVERLIGHT Daten Templates an. Dies sind Schablonen mit deren Hilfe beliebige Objekte durch (mehrere) UI Komponenten angezeigt werden können.

Algorithmus 6 Beispiel für ein Daten Template

```
<DataTemplate>
    <StackPanel>
        <Image Source="{Binding Logo}"/>
        <TextBlock Text="{Binding Text}"/>
    </StackPanel>
</DataTemplate>
```

Wird das im Beispiel 6 definierte Template in einer ListBox verwendet, dessen ItemsSource auf eine Sammlung von passenden Objekten gesetzt ist, dann wird für jedes Objekt ein Daten-Container erstellt, dessen Inhalt durch das Template vorgegeben ist.

Wie im Beispiel bereits angedeutet, können diese Templates beliebig komplex sein. Der Kontext von Datenbindungen innerhalb dieser Templates, ist dabei das Objekt, für welches das Template instanziiert wurde.

4.4 Ressourcen

In jedem Projekt tauchen Teile auf, die an vielen Stellen wieder verwendet werden, z.B. Bilder, Texte oder Vorlagen für bestimmte Stile. Ähnlich dem Konzept der Datenbindungen können in SILVERLIGHT Ressourcen einfach definiert und verwendet werden.

Jedes Steuerelement in SILVERLIGHT kann eine Liste von Ressourcen für sich und seine Kind-Elemente definieren. Außerdem können anwendungsweite Ressourcen definiert werden - siehe Beispiel 7.

Algorithmus 7 Beispiel Ressourcendefinition

```
<Application.Resources>
    <System:String x:Key="AppTitle">
        Open Street App
    </System:String>
</Application.Resources>
```

In Beispiel 4 wurde bereits aufgezeigt, wie auf eine Ressource innerhalb einer Datenbindungs-Definition verwiesen wird. Bei der Auflösung des Namens einer Ressource wird der Elementbaum von unten nach oben durchsucht, bis schlussendlich die anwendungsweiten Ressourcen verwendet werden.

In den bisherigen Beispielen erfolgte der Verweis auf eine Ressource immer mittels {*StaticResource*}. Wie bereits erwähnt existiert erwartungsgemäß auch eine {*DynamicResource*}, die aber erst mit SL4 verfügbar sein wird. Der Unterschied besteht im Ladeverhalten:

Statische Ressourcen werden nur einmal beim Initialisieren des Elements geladen. Sollte sich also das, worauf sich die Ressource bezieht, verändern, würde das die Bindung nicht widerspiegeln. Ein dynamischer Ressourcenverweis hingegen funktioniert genau wie eine Datenbindung.

4.5 Besonderheiten auf dem Handy

Die Entwicklung auf Mobilgeräten unterliegt zum Teil gänzlich anderen Anforderungen, als die Entwicklung für den Desktopbereich. Diese Unterschiede sind größtenteils nicht WP7 spezifisch, müssen hier aber dennoch erwähnt werden, da WP7 einige individuelle Lösungsansätze bietet.

4.5.1 Performance

Während auf “normalen” PCs sowohl Rechenleistung als auch Arbeitsspeicher im Überfluss vorhanden sind, müssen sich Entwickler im mobilen Bereich über diese Aspekte wieder Gedanken machen. Nur dann kann eine ausreichende Performance auf dem Gerät erzielt werden.

Dieses Ziel lässt sich auf zwei verschiedene Arten erreichen: Entweder schränkt man sich soweit ein, dass die Performance des Gerätes ausreicht oder man beein-

flusst die wahrgenommene Performance. Letzteres beschreibt den Effekt, dass einem Anwender eine App die ihn warten lässt, bis alle Daten geladen sind, sehr viel langsamer erscheint, als eine App, die ihm während des Ladevorgangs zumindest schon einzelne Daten anzeigt. Und das obwohl diese meistens insgesamt sogar mehr Zeit benötigt als erstere. Um die wahrgenommene Performance zu verbessern, gibt es abhängig von der Situation verschiedene Möglichkeiten:

Wenn eine größere Datenmenge über eine langsame Handy-Verbindung übertragen werden soll, bietet es sich an, die Daten in kleinen Blöcken zu laden, und erst wenn der Benutzer an das Ende der Liste gescrollt hat, den Ladevorgang fortzusetzen.

Sobald sich die Daten im Speicher befinden steht man allerdings vor dem Problem, dass die Scroll-Performance stark leidet - besonders bei komplexen Daten-Templates. SILVERLIGHT hat für dieses Problem bereits eine eingebaute Lösung - "UI Virtualisierung". Dabei werden die Daten-Container der Templates, die aus dem sichtbaren Bereich heraus-gescrollt wurden, wiederverwendet. Somit wird aufwendiges Zuweisen von neuem Speicher vermieden und es müssen nicht Container für alle Daten-Elemente erzeugt werden.

Falls diese Technik nicht ausreicht um ein flüssiges Scrollen zu ermöglichen, z.B. aufgrund zu komplexer Daten-Templates, existiert auch dafür eine Lösung. In einem erweiterten *ListBox* Steuerelement[8] ist es möglich ein alternatives, vereinfachtes Data-Template anzugeben, das verwendet wird, während der Benutzer scrollt. So kann z.B. während des Scrollens nur der Name eines Films angezeigt werden und sobald die Liste anhält, zusätzlich ein Bild und weitere Informationen.

4.5.2 TOMBSTONING

Eine Konsequenz der geringen Ressourcen ist das so genannte TOMBSTONING. WP7 unterstützt kein Multitasking[9], wie man es auf dem Desktop gewöhnt ist. Stattdessen wird ein Programm, sobald es inaktiv wird, aus dem RAM entfernt. Bevor dies geschieht wird der App zuvor die Möglichkeit gegeben ihren Zustand zu sichern.

Dies geschieht an zwei Stellen:

Zum einen kann jede Seite lokale Daten halten. Da Seiten bei jedem Aufruf neu erstellt werden, müssen diese Daten, für eine eventuell später ausgeführte "Zurück" Navigation, bei jedem Verlassen gespeichert werden. Dies geschieht durch

[8] Das benötigte Steuerelement ist nicht Teil der Standardbibliothek und kann hier gefunden werden: `http://goo.gl/14H2U`

[9] Das OS selbst unterstützt Multitasking und ein zukünftiges Update soll es auch Apps zugänglich machen.

setzen eines STATE Dictionaries, auf welches der Navigationsservice Zugriff hat, der die Daten dann serialisiert.

Zum anderen kann die *App* Klasse globale Daten halten. Diese können entweder mit einer Dictionary basierten Methode gespeichert werden oder manuell auf dem normalen Dateisystem.

Wenn eine Anwendung oder eine Seite keinen inhärenten Zustand besitzt, ist die Verwendung dieser Methoden aber nicht verpflichtend. Sie werden dann einfach neu erstellt, wie bei ihrem ersten Aufruf.

4.5.3 UI Design

Ein weiterer Aspekt, auf den hier aber nicht näher eingegangen werden soll, ist die Bauform. Durch sie bedingt steht sehr viel weniger Bildschirmfläche zur Verfügung, was u.U. ein komplett unterschiedliches Design nötig machen kann. Außerdem sind in diesem Bereich Eingabemethoden üblich, die auf dem Desktop oft gar nicht zur Verfügung stehen. (Z.B. Multi-Touch-Screens, Beschleunigungs- und Lagesensoren)

4.6 Entwicklungsumgebung

Microsoft stellt sowohl VISUAL STUDIO 2010 als auch EXPRESSION BLEND in einer speziellen Version für WP7 kostenlos zur Verfügung. VISUAL STUDIO richtet sich dabei an Entwickler und BLEND an Designer. Aber prinzipiell könnte man beide Tools für die gesamte Entwicklung verwenden.

Über beide Tools könnte man ganze Bücher schreiben, was aber ganz klar den Rahmen dieser Arbeit sprengen würde. Allerdings gibt es einen Punkt den man als Entwickler für WP7 beachten muss:

Der Emulator der im WP7 SDK mitgeliefert wird, verhält sich nicht 100%ig wie ein echtes Gerät. Da der Emulator-Code auf x86 portiert wurde ist die Performance in machen Teilen besser, und in anderen Teilen schlechter. Manches Verhalten lässt sich nur auf einem von beiden reproduzieren und H.264 Videos können nur auf echten Geräten wiedergegeben werden.

Ein weiterer Punkt, der nirgends dokumentiert ist, und somit im Rahmen dieser Arbeit erstmals aufgedeckt wurde, ist ein Unterschied in den SSL Zertifikaten. Während auf dem Gerät alle üblichen Zertifikate installiert sind, fehlt auf dem Emulator zumindest das von DIGICERT. Das verursacht Verbindungsabbrüche an HTTPS-Verbindungen, die auch ohne wie auch immer geartete, hilfreiche Fehlermeldung eine Ausnahme werfen, falls ein entsprechendes Zertifikat verwendet wird.

Aus diesem Grund sollte man stets auf der Zielhardware testen - wobei möglichst mehr als nur ein Gerät verwendet werden sollte.

5 Fazit - Erfolgschancen von WP7

Zum heutigen Tag hat der WP7 Marketplace über 17.000 Apps und mehr als 36.000 Personen haben sich für das Entwicklerprogramm registriert.[7] Seit einem Monat rollt Microsoft das erste große Update für WP7 aus. Dieses enthält *Copy&Paste* als auch viele Performance- und Stabilitäts-Verbesserungen. Außerdem wurde das "Mango" Update vorgestellt, dass im Herbst für alle WP7 Geräte zur Verfügung stehen soll und endgültig mit Android und iOS in Sachen Funktionsumfang gleichziehen soll.

Ein weiterer Faktor, der die Zukunft von WP7 mit Sicherheit entscheidend beeinflussen wird, ist die Allianz mit Nokia, die diesen Monat endgültig beschlossen wurde.

Auch wenn Microsoft bis heute keine offiziellen Zahlen vorgelegt hat, wie viele *Geräte* tatsächlich verkauft wurden, sind diese Fakten ein starker Beleg dafür, dass sich WP7 noch lange am Markt halten wird.

Noch ist die größte Schwäche von WP7, dass es im Gegensatz zu Windows Mobile 6.5 (heute "Windows Phone Classic") noch nicht für Geschäftskunden tauglich ist. Da der Business-Sektor traditionell Microsofts stärkster Geschäftsbereich ist, muss dieses Manko so schnell wie möglich beseitigt werden. Dann steht dem Erfolg von *Windows Phone 7* nichts mehr im Weg.

Literatur

[1] F. Rapp, "Entwicklung der Open-Street-Applikation für Windows Phone 7," *Universität Ulm*, 2010.

[2] W. Rolke, Abrufdatum: 02.05.2011. [Online]. Available: http://www.wolfgang-rolke.de/wince/platform.htm

[3] S. Guthrie, "Silverlight," Abrufdatum: 02.05.2011. [Online]. Available: http://weblogs.asp.net/scottgu/archive/2007/05/07/silverlight.aspx

[4] M. de Icaza, "Moonlight 4 preview 1 is out," Abrufdatum: 02.05.2011. [Online]. Available: http://tirania.org/blog/archive/2011/Feb-16.html

[5] MSDN, “Contrasting Silverlight and WPF,” Abrufdatum: 19.05.2011. [Online]. Available: http://msdn.microsoft.com/en-us/library/ff921107(v=pandp.20).aspx

[6] D. Beck, “WPF & Silverlight,” *Universität Stuttgart*, 2008.

[7] W. P. Applist, “Wp7 app statistics.” [Online]. Available: http://www.windowsphoneapplist.com/stats/