Entwicklung auf Windows Phone 7

Am Beispiel einer OpenStreetMap Anwendung

Martin Rauscher

5. April 2011

1 Einführung

Laber laber laber

1.1 Vorstellung der Beispielanwendung

Im Rahmen dieser Arbeit wurde eine Anwendung erstellt, die das Betrachten von Karten von OpenStreetMap (und anderen Anbietern) ermöglicht. Sie ermöglicht die Steuerung über die üblichen Multi-Touch-Gesten, das suchen nach Orten und das erstellen von Favoritenlisten. Desweiteren können Routen geplant werden.

Alle Beispiele in dieser Arbeit sind aus dem Code dieser Anwendung. Für nähere Informationen zur "OpenStreetApp" wird auf XXXXX verwiesen.

1.2 Rahmen dieser Arbeit

Um Anwendungen für WP7 zu schreiben gibt es grundsätzlich zwei Möglichkeiten[1]:

1. Entwicklung auf Basis von Silverlight
2. Nutzung des XNA Framework - eine gekapselte DirectX-Schnittstille - welches hauptsächlich für die Entwicklung von Spielen verwendet wird.

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

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

2 Windows Phone 7

Windows Phone 7 ist hinsichtlich der Benutzeroberfläche als auch der Anwendungsplattform ein 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.

Gründe nicht auf das bestehende Windows Mobile aufzusetzen waren [XXXX]

- UI-Konzept zu stark Desktoporientiert
- Fehlende Multi-Touch Unterstützung
- Entwicklung moderner Anwendungen schwierig

WP7 begegnet diesen Problemen mit einer starken Unterstützung für Multi-Touch in Silverlight und den integrierten Kontrollelementen, die es leicht machen sollen, moderne Anwendungen zu schreiben, die sich in das UI Konzept einfügen.

3 Silverlight und die Windows Presentation Foundation

Das 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 was die groben Unterschiede sind.

3.1 Windows Presentation Foundation (WPF)

Ende 2006 stellte Microsoft die finale Version 3.0 des .NET Framwork vor. Mit diesem wurde zum ersten mal die, bis dahin unter dem Codenamen Avalon entwickelte, Windows Presentation Foundation (WPF) der Öffentlichkeit zugänglich gemacht. Microsoft hoffte das 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 Abwendung von GDI(+) für das GUI-Rendering. Mit WPF wird erstmals voll 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 (unter dem Codenamen Windows Presentation Foundation/Everywhere (WPF/E) entwickelt) vor. Silverlight ist, plakativ gesprochen, eine stark reduziert Version des .NET Framework und eine Untermenge[2] der WPF Funktionalität.

Auch wenn Silverlight oft als Konkurenz zu Adobes Flash Player gesehen wird, wurde es hauptsächlich mit dem Ziel entwickelt so genannte Rich Internet Applications (RIAs) zu ermöglichen, die hauptsächlich im Geschäftsbereich ihren Einsatzzweck haben. 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: U.a. wurden die Olympischen Winterspiele von 2008 und einige amerikanische Sportgroßereignisse exklusiv via Silverlight gestreamt; außerdem setzt das deutsche Videoportal “Maxdome” auf Silverlight.

Silverlight wurde vom Grund auf Plattformunabhängig konzipiert. Microsoft stellt es für die meisten Windows Versionen sowie für Mac OS X zur Verfügung. Desweiteren ist ein Großteil der Spezifikationen und einige Teile des Codes öffentlich zugänglich. Darauf aufbauend entwickelt das Mono Projekt unter Leitung von Novell einen Linux Port. “Moonlight” ist momentan Kompatibel mit Silverlight 3.

3.3 Unterschiede

nochmal

Mit dem Ziel der Plattformunabhängigkeit gingen natürlich auch höhere Ansprüche an einen geringen Ressourcenverbrauch einher. Das hatte zur Folge, dass einige Funktionen - z.B. Hardwarebeschleunigung und einige Controls - erst in späteren Versionen zur Verfügung standen. Bis jetzt sind noch einige Konzepte, wie z.B. Trigger und implizite Styles, noch immer nur in WPF verfügbar.

[2] Aufgrund von unterschiedlichen Releasezyklen hat die aktuelle WPF einige Klassen nicht, die in Silverlight enthalten sind. Dies 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.

4 Entwicklen mit Silverlight

4.1 Struktur einer Silverlight Anwendung

Eine Silverlight Anwendung wird immer in Form einer in "*.XAP" umbenannten ZIP-Datei 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
- Die Ressourcen (Bilder, Töne, etc.)

Wenn das Betriebssystem eine App startet, wird zuerst das Manifest eingelesen und daraus ein xxxxxxx

navigation

4.2 XAML

XAML ist eine deklarative Sprache zur Beschreibung von Datenstrukturen die insbesondere in 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 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 Kontext 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 die Eigenschaften u.U. ohne Bedeutung.

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 den komplexen Attributdefinitionen wird der Name des übergeordneten Elements verwendet um die Herkunft des Attributs zu qualifizieren. Der Wert des Attributs wird dabei in einem statischen Attribut der Klasse des übergeordneten Elements gespeichert.

4.2.3 Markup Extensions

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

Algorithmus 3 Beispiel für eine Markup Extension

```
<TextBlock Text="{StaticResource␣AppName}" />
<!-- oder -->
<TextBlock Text="{StaticResource␣ResourceKey=AppName}" />
<!-- vs. -->
<TextBlock>
        <TextBlock.Text>
                <StaticResource ResourceKey="AppName" />
        </TextBlock.Text>
</TextBlock>
```

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".

Silverlight bietet - in der aktuellen Version - keine Möglichkeit eigene Markup Extensions zu definieren. Es sind nur die folgenden drei vordefiniert: Binding, StaticResource, DynamicResource. Auf deren Bedeutung wird im folgenden Abschnitt eingegangen.

4.3 Datenbindung

Ein 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 keine 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 con
```

xxxx Diese Eigenschaft enthält den aktuellen Kontext für Datenbindungen. In Beispiel 4 kann man sehen wie die Definition einer Datenbindung mit Hilfe der {Binding} Markup Extension aussieht. Dabei bezieht sich die Datenbindung auf das Objekt das der aktuelle Datenkontext ist, da nichts anderes angegeben wurde. Dieser Kontext ergibt sich aus dem Element innerhalb dessen das Binding definiert

wurde. Jedes GUI-Element[4] hat eine DataContext Eigenschaft, die es an eventuell vorhandene Kind-Elemente weitervererbt.

4.3.1 Dependency Properties

Ein Problem vieler Datenbindungs-Ansätze ist, dass sie Reflection[5] verwenden, da die Eigenschaften oft als String angegeben werden. Außerdem muss jede Klasse typischerweise ein Interface implementieren, dass die eventuellen Ziele einer Datenbindung benachrichtigt.

In Silverlight wird dieses Problem mit so genannten Depenency Properties (DPs) gelöst. Im Grunde sind DPs Dictionaries die als statische Eigenschaften in der Klasse definiert werden. Dieses weißt dann einem Objekt einen Wert zu.

Der große Vorteil von DPs ist, dass sich beliebige Objekt 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 einfach Standardwerte definieren und automatisches Vererben von Werten auf Kind-Objekte realisieren.

4.3.2 Converters

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 dem die Konvertierung natürlich automatisch passiert, ist es oft komplizierter.

[4] Genauer, jede von FrameworkElement erbende Klasse.

[5] 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.

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;
   }
}
```

Die Lösung dieses Problems sind so genannte Converter, die beliebige Typen in einander konvertieren können. Converter sind Klassen, die 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.

4.4 Resourcen

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 List von Ressourcen für sich und seine Kindelemente definieren. Außerdem können Anwendungsweite Ressourcen definiert werden - siehe Beispiel 6.

Algorithmus 6 Beispiel Resourcendefinition

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

In Beispiel 4 kann man sehen, wie auf eine Resource innerhalb einer Datenbindungs-Definition verwiesen wird. Bei der Auflösung des Namens einer Ressource wird der Elementbaum von unten nach oben durchsucht, bis schlussendlich noch die anwendungsweiten Ressourcen durchsucht werden.

In den bisherigen Beispielen erfolgte der Verweise auf eine Ressource immer mittels {StaticResource}. Erwartungsgemäß existiert auch eine {DynamicResource}. Der Unterschied besteht im Ladeverhalten:

Statische Ressourcen werden nur ein mal beim Initialisieren des Elements geladen. Sollte sich also das worauf sich die Ressource bezieht verändern, würde das die Bindung nicht wiederspiegeln. 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 Übermaß vorhanden ist, müssen sich Entwickler im mobilen Bereich über diese Aspekte wieder Gedanken machen. Nur dann kann eine ausreichende Performance auf dem Gerät zu erzielt werden.

Beispiele im Vergleich WPF SL/WP7

lazy loading

ui virtualisierung

4.5.2 Tombstoning

Eine Konsequenz der geringen Ressourcen ist das so genannte Tombstoning. WP7 unterstützt kein Multitasking[6], 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 Page lokale Daten halten. Da Pages bei jedem Aufruf neu erstellt werden, müssen diese Daten, für eine eventuell später ausgeführte "zurück" Navigation, bei jedem Verlassen gespeichert werden. Dies geschieht durch setzen eines State Dictionaries, auf das der Navigationsservice Zugriff hat.

Zum Anderen kann die App Klasse globale Daten halten. Diese können entweder auch mit einer Dictionary basierten Methode gespeichert werden, oder aber auch manuell mit dem normalen Dateisystem.

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

4.5.3 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 von Nöten machen kann. Außerdem sind in diesem Bereich Eingabemethoden üblich, die auf dem Desktop oft garnicht zur Verfügung stehen. (Z.B. Multi-Touch-Screens, Beschleunigungs- und Lagesensoren)

4.6 Entwicklungsumgebung

Visual Studio

5 Fazit - Erfolgschancen von WP7

noch-probleme

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

Literatur

[1] Erste Quelleden