Kapitel 3

Windows Phone unter der Haube

In diesem Kapitel:

Patrick Getzmann, Simon Hackfort, Peter Nowak: Entwickeln für Windows Phone 7.5. Architektur, Frameworks, APIs. Microsoft Press 2011 (ISBN 978-3-86645-465-1)

Die neue Hardwareplattform

Eine der vielen Neuerungen, die Windows Phone mitbringt, ist die Hardwarespezifikation für Hersteller, die Windows Phone auf ihren Geräten einsetzen wollen.

An und für sich kann es dem Entwickler relativ egal sein, auf was für einer Hardware seine Applikation läuft, solange sie den Anforderungen der jeweiligen Applikation entspricht. Doch genau an diesem Punkt gab es in der Vergangenheit des Öfteren Probleme. Die Gerätehersteller waren im Prinzip vollkommen frei bei der Auswahl der Hardware.

In der Anfangszeit der Pocket PCs musste man sich als Entwickler noch mit mehreren Prozessorarchitekturen wie MIPS, ARM, SH2/4 oder x86 befassen. Im Regelfall war die jeweilige Prozessorarchitektur für den eigentlichen Programmcode relativ uninteressant und man brauchte nichts weiter zu beachten. Von Nachteil war jedoch, dass man für jeden Prozessortyp eine eigene Projektausgabe erhielt, die dementsprechend separat in eigenen CAB-Dateien oder in anderweitiger Form bereitzustellen waren. Dieses Problem hatte sich zumindest für reine Pocket PC- beziehungsweise Windows Mobile-Entwickler bald in Luft aufgelöst, denn Windows Mobile 2003 gab es nur noch in der ARM-Architektur.

Dafür brachten die Geräte immer mehr zusätzliche Hardwarekomponenten mit sich. Bluetooth und WLAN waren zwei typische Vertreter derartiger Komponenten. Für Bluetooth standen diverse Schnittstellen zur Verfügung, die leider nicht zueinander kompatibel waren. WLAN war auch sehr problematisch, da zu dieser Zeit noch jeder Hersteller sein eigenes Süppchen kochte und der NDIS-Standard für WLAN mehr einer optionalen Richtlinie glich, und weniger einem Konzept, auf das sich ein Entwickler verlassen konnte. Auch wenn dies eher Randgebiete der Entwicklung waren und nur wenige Entwickler auf solche Zusatzhardware zugreifen mussten, gab es für die meisten Entwickler auch so genug Probleme.

Hierzu gehörte zum einen die generelle Unterscheidung zwischen Geräten mit Touchscreen und den Geräten ohne Touchscreen, bei denen sich die Steuerung ziemlich unterschied. Dazu gab es eine nahezu unüberschaubare Anzahl an Formfaktoren und Bildschirmauflösungen, die es in der Applikation zu berücksichtigen galt, wenn man nicht gerade für ein explizites Gerät entwickelt hat.

Mit der Zeit gab es dann immer mehr neuere Hardwarekomponenten wie integriertes GPS oder Sensoren, die Bewegungen oder Helligkeit auswerten konnten. War GPS durch die Einführung der GPS-API noch relativ einfach zu unterstützen (von wenigen »eigenwilligen« Treiberimplementierungen einiger Hersteller einmal abgesehen), waren Sensoren schon etwas heikler. Es gab keine einheitliche API. Diverse Chiphersteller oder verschiedene ROM-Versionen zum gleichen Chip zeigten ein unterschiedliches Verhalten. Zusätzlich dazu waren auch nicht immer alle Gerätehersteller auskunftsfreudig und offizielle SDKs zu solchen Systemkomponenten suchte man meist vergeblich. Man musste auf so genannte Leaks, also versehentliche Veröffentlichung von Dokumenten, oder auf direktes Reverse-Engineering zurückgreifen. Wollte man solche Funktionen eines Geräts benutzen, befand man sich in einer Grauzone, ohne Garantie, dass die Software beim nächsten ROM-Update noch funktionieren würde. Ohne genaue Kenntnis der Wirkungen von Funktionsaufrufen konnte man gegebenenfalls leicht Schaden verursachen. Zusätzlich erschwerte solch »spekulativer« Code auch die Wartbarkeit und Pflege eines Projekts. Man stelle sich nur mal Code vor, der auf drei Geräten die Bewegungssensoren abfragen sollte, die in unterschiedlichen ROM-Versionen auch noch unterschiedlich angesprochen werden mussten. Und wenn man den damit verbundenen Test- und Bereitstellungsaufwand betrachtet, war man mit der Unterstützung solcher Zusatzhardware ziemlich gut beschäftigt.

Um diesen Wildwuchs zu bekämpfen und den Entwicklern die Implementierung zu vereinfachen, wurde für Windows Phone eine minimale Hardwareanforderung seitens Microsoft an die Gerätehersteller eingeführt.

Patrick Getzmann, Simon Hackfort, Peter Nowak: Entwickeln für Windows Phone 7.5.
Architektur, Frameworks, APIs. Microsoft Press 2011 (ISBN 978-3-86645-465-1)

Dank dieser Hardwareanforderung gibt es nun eine verlässliche Basis, auf die der Entwickler in seiner Applikation aufsetzen kann. Doch die Verfügbarkeit dieser Basishardware ist noch nicht alles, denn die Spezifikation sieht auch vor, dass sich die Hardware an definierte Richtlinien hält und vor allem auch über einheitliche Schnittstellen verfügen muss, die dem Anwendungsentwickler über die Windows Phone-API bereitgestellt werden. Die gemeinsame Basis erleichtert die Implementierung hardwarespezifischer Funktionen ungemein und kommt vor allem letztlich auch dem Endanwender der Applikation zugute.

Die folgenden Abschnitte geben einen Einblick in diese minimalen Hardwareanforderungen.

Display & Auflösung

Im Gegensatz zu den bisher im Windows Mobile-Umfeld üblicherweise verwendeten resistiven Touchscreens, die auf Druck reagieren, werden Windows Phone-Geräte generell nur mit kapazitiven Touchscreens ausgeliefert. Kapazitive Touchscreens reagieren nicht auf Druck, sondern mittels elektrischer Feldstärkenmessung. Berührt ein Finger das Display, wird das ruhende statische elektrische Feld verändert. Ein klarer Vorteil ist die bessere Fingerbedienbarkeit sowie einfache Multi-Touch-Unterstützung. Die Eingabe mit den bisher üblichen Stiften ist nicht mehr möglich. Allerdings sind mittlerweile spezielle Stifte für kapazitive Bildschirme erhältlich. Jedes Windows Phone unterstützt Multi-Touch und wird mindestens 4 Berührungspunkte bieten. Die Auflösung des Bildschirms beträgt derzeit bei Windows Phone 800×480 Pixel.

Prozessor & Grafik

Windows Phone arbeitet auf aktuellen ARMv7-Prozessoren mit mindestens 1 GHz Rechenleistung. Die Grafikchips müssen kompatibel zu DirectX 9 sein und hardwareseitige Beschleunigung für DirectX und Direct3D unterstützen. Dadurch ist eine hohe Performance für XNA-Spiele und für grafisch aufwändige Silverlight-Anwendung gegeben.

Speicher

Grafisch aufwändige Anwendungen und Spiele verbrauchen viel Speicher. Ein Windows Phone kann jedoch eine große Anzahl von E-Mails und über die Zune-Integration Unmengen von Video- und Audiodateien beherbergen.

Um diese Datenmengen auch in einer minimalen Hardwarekonfiguration speichern zu können, sind die Anforderungen hier schon relativ hoch angesiedelt. Der Arbeitsspeicher ist auf mindestens 256 MB festgelegt. Der Datenspeicher ist mit mindestens 8GB hoch bemessen. Da austauschbare Speicherkarten nicht vorgesehen sind, sollte vor dem Erwerb der Hardware der benötigte Speicherbedarf für die persönlichen Ansprüche genau ermittelt werden.

Tastatur & Bedienelemente

Für viele Benutzer ist eine Hardwaretastatur ein wichtiges Kaufargument. Einigen Benutzern ist ein Gerät mit Tastatur allerdings zu schwer und zu groß. Andere Benutzer wiederum wollen beim Schreiben von E-Mails oder SMS nicht auf eine Tastatur verzichten. Daher sieht die Windows Phone-Hardwarespezifikation eine Tastatur nur als optionale Komponente vor.

Patrick Getzmann, Simon Hackfort, Peter Nowak: Entwickeln für Windows Phone 7.5.
Architektur, Frameworks, APIs. Microsoft Press 2011 (ISBN 978-3-86645-465-1)

Eine solche Hardwaretastatur unterliegt jedoch auch einigen Einschränkungen. Verwendungszweck ist lediglich das Schreiben von Text. Die Benutzung der Tastatur zum Steuern des Geräts, beispielsweise innerhalb von Applikationen, ist nicht vorgesehen.

Um eine konsistente Bedienung auch über verschiedene Geräte hinweg zu gewährleisten, sieht die Hardwarespezifikation eine Reihe von Tasten vor, die sich auf jedem Windows Phone wiederfinden lassen:

Die Powertaste

Die Powertaste dient zum Ein- und Ausschalten des Geräts, beziehungsweise auch des Bildschirms. Je nach Gerätezustand, teilweise auch der Haltedauer der Taste, wird eine andere Aktion ausgeführt.

Die Lautstärketasten

Die Lautstärketasten dienen grundsätzlich zum Regeln der systemweiten Gerätelautstärke. Sie sind darüber hinaus auch dann noch aktiv, wenn im Hintergrund Musik abgespielt wird und der Bildschirm gesperrt ist. Bei einem eingehenden Telefonanruf kann über die Lautstärketasten der Klingelton vorübergehend ausgeschaltet werden.

Die Kamerataste

Die Kamerataste wird zum Starten der Kameraanwendung verwendet. Die Taste kann komplett oder nur zur Hälfte durchgedrückt werden. Während ein komplettes Drücken der Taste eine Aufnahme startet, kann über »halbes« Drücken der Autofokus aktiviert oder deaktiviert werden.

Die Kamerataste kann auch im Standby- oder gesperrtem Modus verwendet werden und ermöglicht es dem Anwender, jederzeit einen Schnappschuss aufzunehmen.

Die Zurück-Taste

Die Zurück-Taste wird zur Navigation innerhalb einer Anwendung und zwischen verschiedenen Anwendungen verwendet. Das zugehörige Ereignis zur Taste kann in der eigenen Applikation abgefangen werden, um beispielsweise Seitenübergänge zu animieren oder Menüs zu schließen.

Die explizite Implementierung der Zurück-Taste darf auf keinen Fall das eigentliche Navigationsmodell des Telefons behindern oder außer Kraft setzen.

Die Starttaste

Über die Starttaste gelangt der Anwender von jeder beliebigen Applikation aus in den *Start* des Geräts.

Die Suchtaste

Über die Suchtaste kann der Anwender von jeder beliebigen Applikation aus die Bing-Websuche starten – abgesehen von einigen wenigen Ausnahmen. In der Zune-Applikation wird die Zune Marketplace-Suche gestartet, im Bereich der Kontakte lassen sich die Kontakte durchsuchen sowie im Mail-Bereich die E-Mails.

Patrick Getzmann, Simon Hackfort, Peter Nowak: Entwickeln für Windows Phone 7.5.
Architektur, Frameworks, APIs. Microsoft Press 2011 (ISBN 978-3-86645-465-1)

Audio

Ganz ungeachtet der Tatsache, dass ein Windows Phone ja auch ein Telefon ist, wird Audio in vielen Applikationen genutzt. Doch nicht nur die Audioausgabe ist wichtig, etwa für Hinweise auf Ereignisse wie eingehende Telefonanrufe, Termine oder auf sonstige Applikationsereignisse und das Auslösen von Schaltflächen.

Auch die Audioeingabe ist wichtig. Zum einen ist sie selbstverständlich für ein Telefonats notwendig. Zum anderen bieten sich über die Spracheingabe auch vollkommen neue Steuerungsmöglichkeiten für Applikationen an. So könnte eine Applikation komplett über Sprachbefehle gesteuert werden. Auch können definierte Geräuschpegel verwendet werden, wie sie etwa beim Pusten in das Mikrofon entstehen.

Für Sie als Applikationsentwickler können also durchaus beide vorhandenen Audioelemente des Geräts interessant sein. Daher sind beide Schnittstellen, sowohl das Mikrofon als auch der Lautsprecher, über APIs ansprechbar.

Eine zusätzliche Audioschnittstelle ist der Kopfhöreranschluss. Eigentlich dient der Kopfhörer als zusätzliche Audioausgabe, um beispielsweise Musik oder ein Spiel ohne Beeinträchtigung Dritter genießen zu können.

Allerdings nimmt der Kopfhörer auch eine Sonderstellung ein, nämlich dann, wenn das FM-Radio verwendet werden soll. Das Kabel des Kopfhörers wird in diesem Fall als Antenne für den Empfang des Radiosignals verwendet. Es kann zwar zusätzlich der Lautsprecher zur Ausgabe des Audiosignals aktiviert werden, jedoch wird der Kopfhörer zwingend benötigt, um überhaupt ein Sendersignal zu erhalten.

Sensoren

Moderne Hardware, gerade für Kommunikationsgeräte wie Mobiltelefone, kommt heutzutage nicht mehr ohne Sensoren aus, zumindest dann nicht, wenn sie noch den Erwartungen der Anwender entsprechen soll. Sensoren im weiteren Sinne sind elektronische Bauteile, die bestimmte physikalische Eigenschaften messen können. Dank solcher Sensoren kann ein Gerät etwa auf Bewegung und Drehungen reagieren, die Bildschirmhelligkeit anpassen oder seine Position bestimmen.

Nachfolgend nun ein Überblick der Sensoren, die ein Windows Phone-Gerät mitbringen muss:

Accelerometer

Ein Accelerometer (Beschleunigungssensor) misst eigentlich nur eine reine Beschleunigung, also die Zu- oder Abnahme einer Bewegung. Da aber reine Beschleunigungsinformationen relativ nutzlos wären, sind in modernen Mobiltelefonen häufig vollständige Gravitationssensoren, kurz G-Sensoren, verbaut. Diese messen nicht nur die Beschleunigung, sondern auch die Gravitationsveränderung des Geräts bei Bewegungen.

Daher kann beispielsweise automatisch zwischen Hoch- und Querformat umgeschaltet werden oder innerhalb von Spielen eine Figur über Bewegung des Geräts gesteuert werden.

Als Entwickler erhält man über die Accelerometer-API Zugriff auf diesen Sensor.

Patrick Getzmann, Simon Hackfort, Peter Nowak: Entwickeln für Windows Phone 7.5.
Architektur, Frameworks, APIs. Microsoft Press 2011 (ISBN 978-3-86645-465-1)

Gyroskop

Ein Gyroskop kann optional in Windows Phone 7.5 Geräten enthalten sein. Ein Gyroskop ist ein Kreiselinstrument, wodurch eine präzisere Lagebestimmung als bei einem Accelerometer erreicht werden kann. Für den Entwickler steht unter Windows Phone 7.5 die Gyroscope-API bereit.

A-GPS

Ein A-GPS-Sensor (Assisted Global Positioning System) ist eine Erweiterung der herkömmlichen GPS-Empfänger in mobilen Geräten. Zusätzlich zu den GPS-Informationen können mit A-GPS Systemen auch weiterreichende Techniken wie GSM-Triangulation oder die Standortbestimmung über WLAN-Netze (per IP-Lokalisierung), verwendet werden.

Als Entwickler erhält man über die Location Service-APIs Zugriff auf die Sensorinformationen.

Compass

Ein Kompass dürfte wohl jedem bekannt sein. In diesem Fall ist es ein digitales Messinstrument, das die Bewegung des Geräts relativ zum magnetischen Feld der Erde misst. Ein solcher Kompass-Sensor kann in Windows Phone-Geräten enthalten sein und soll eine präzisere GPS-Positionierung ermöglichen, indem das Gerät die genaue Ausrichtung des Geräts mit dem GPS-Signal abgleichen kann, um eine verbesserte Richtungsdarstellung auf Karten zu gewährleisten. Ein Zugriff auf den digitalen Kompass, über eine entsprechende Entwicklerschnittstelle, ist ab Windows Phone 7.5 möglich.

Proximity

Der Proximity-Sensor (Näherungssensor) wird verwendet, um den Bildschirm bei Telefonverbindungen auszuschalten, sobald die Entfernung eines Objekts zum Sensor, und somit zum Gerät, eine Distanz von 15 Millimetern unterschreitet.

Lichtsensor

Der Lichtsensor misst, wie der Name vermuten lässt, die Umgebungshelligkeit des Geräts. Ein Windows Phone-Gerät verwendet den Lichtsensor für die automatische Helligkeitsregelung des Bildschirms.

HINWEIS Auf den Proximity- und Lichtsensor ist seitens der API kein Zugriff möglich.

Verbindungen

Für Telefongespräche und um stets die neuesten Daten wie E-Mails und Neuigkeiten aus sozialen Netzwerken beziehen zu können, muss ein Windows Phone-natürlich die dementsprechenden Schnittstellen mitbringen. So verfügt jedes Gerät über moderne GSM-Technologie (beziehungsweise für andere Regionen künftig auch CDMA) oder über WLAN, um den jeweils besten Dienst von GPRS bis HSDPA oder vom heimischen Netzwerk bis hin zu öffentlichen Hotspots nutzen zu können.

Zusätzlich bietet die Hardware natürlich auch die Nutzung von Bluetooth. Aktuell gilt dies jedoch nur für die Verbindung zu Headsets.

Patrick Getzmann, Simon Hackfort, Peter Nowak: Entwickeln für Windows Phone 7.5.
Architektur, Frameworks, APIs. Microsoft Press 2011 (ISBN 978-3-86645-465-1)

Sonstige Hardware

Darüber hinaus enthält Windows Phone noch einige weitere Hardware-Elemente. So enthält jedes Gerät ein FM Radio und eine Kamera mit mindestens 5 Megapixel-Sensor inklusive automatischer Fokussierung und Blitzlicht.

Auch der Anschluss für ein Datenkabel beziehungsweise Ladegerät ist vorgeschrieben und wird als MicroUSB standardisiert ausgeführt – was sicherlich jedem Entwickler oder Besitzer mehrerer Geräte entgegenkommt.

Fazit

Wie bei einzelnen Punkten bereits erwähnt, hat der Entwickler nicht auf alle Hardwarebereiche Zugriff. Dies kann sich natürlich in Zukunft noch ändern. Auch ist davon auszugehen, dass sich künftig auch die Hardwareanforderungen ändern werden, wenn etwa ein neuer Sensortyp Verbreitung finden sollte, von dem die Windows Phone-Plattform profitieren kann. Wie jedoch die genauen Prozesse dabei in Bezug auf SDK-Updates, Geräteupdates und vor allem auch die Abwärtskompatibilität zu älteren Geräten aussehen werden, ist derzeit noch nicht bekannt.

Auf für Entwickler zugängliche Hardwarekomponenten wird detailliert in Kapitel 6 »Arbeiten mit plattformspezifischen APIs« eingegangen.

Die neue Softwareplattform

Eine neue Hardware allein macht allerdings noch keine neue Plattform aus. Daher hat sich auch die Softwareplattform grundlegend geändert – sie wurde vollständig neu erdacht und entwickelt.

Die nachfolgende Grafik zeigt einen Überblick über die neue Windows Phone-Plattform. Auf die Details der einzelnen Bereiche wird im Laufe des Kapitels näher eingegangen.

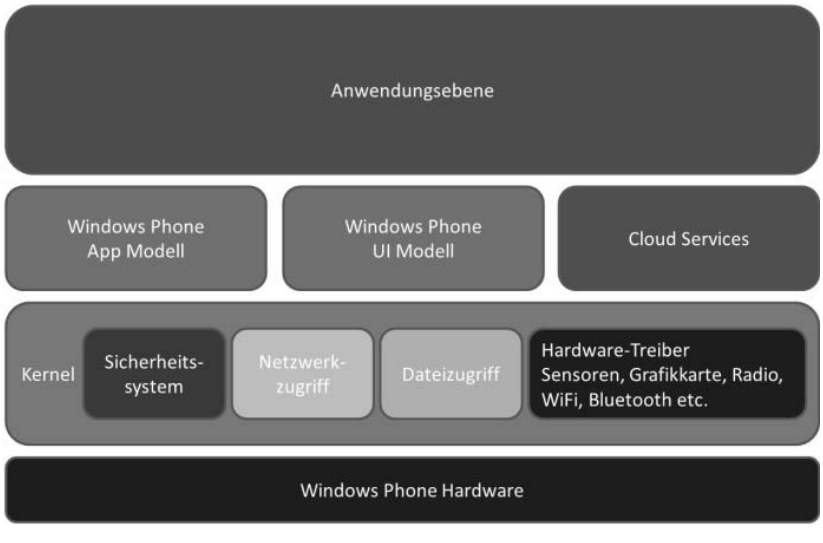

Abbildung 3.1 Übersicht über die neue Softwareplattform

Patrick Getzmann, Simon Hackfort, Peter Nowak: Entwickeln für Windows Phone 7.5.
Architektur, Frameworks, APIs. Microsoft Press 2011 (ISBN 978-3-86645-465-1)

Die Anwendungsschicht

Die Anwendungsschicht ist die sicherlich interessanteste Schicht der neuen Architektur. Sie ist die einzige Schicht, die für den Zugriff durch Anwendungsentwickler, also Sie, vorgesehen ist.

Im Vergleich zu Windows Mobile ist es mit Windows Phone nicht mehr möglich, eigene Treiber oder Dienste auf dem Gerät zu installieren. Als Windows Phone-Entwickler hat man nur noch Zugriff auf einen *Usermodus*. Doch selbst dieser ist stark reglementiert. So hat man keinen direkten Zugriff auf Funktionen wie das Telefon, E-Mail oder die Kalenderdaten des Anwenders. Auch gibt es keinen Zugriff auf die Systemeinstellungen. Dies scheint zwar zunächst eine erhebliche Einschränkung für den Entwickler zu sein und dürfte gerade für erfahrene Windows Mobile-Entwickler ziemlich gewöhnungsbedürftig wirken. Doch diese applikationsseitigen Einschränkungen werden sich letztendlich positiv für den Anwender auswirken. Denn dieser steht bei Windows Phone immer im Vordergrund.

So wird es für viele Funktionen, die von Applikationen initiiert werden, wie das Versenden einer SMS oder das Tätigen eines Telefonanrufes, nur noch API-Aufrufe geben, die Systemdialogfelder nach sich ziehen. Damit ist sichergestellt, dass der Anwender immer einbezogen wird und ein konsistentes Verhalten der Windows Phone-Plattform gewährleistet ist, auch über Geräte unterschiedlicher Hersteller und verschiedene Applikationen hinweg.

Das globale Ändern von Systemeinstellungen wie Lautstärke oder Farbschemata ist aus einer Anwendung heraus nicht möglich, da die Applikationen vollständig isoliert ausgeführt werden. Diese Isolation betrifft nicht nur die verfügbaren APIs, sondern auch Dateizugriffe.

Diese vollständige Isolation einer Anwendung in Kombination mit den eingeschränkten Funktionen beziehungsweise Reichweiten der APIs hilft aber nicht nur dem Anwender, nahezu vollständige Kontrolle über das Geschehen auf seinem Gerät zu behalten, sondern es steigert auch die allgemeine Sicherheit und die Reaktionszeiten auf der Windows Phone-Plattform.

Um diese Prinzipien konsequent umsetzen zu können, wurde auch die Entwicklerebene im Vergleich zu Windows Mobile komplett neu gestaltet.

So ist Windows Phone aus Entwicklersicht ein reines *managed* System. Das bedeutet, es kann nur noch .NET Compact Framework Code ausgeführt werden. Das bisher im .NET Compact Framework weit verbreitete *Plattform Invoke*, also das Laden und Verwenden von Funktionen aus nativen C/C++ DLLs, kann nicht mehr verwendet werden.

Microsoft hat sich allerdings nicht nur von der nativen Entwicklungsmöglichkeit getrennt, sondern zugleich zwei mehr oder weniger neue Frameworks für die Anwendungsentwicklung eingeführt.

Das .NET Compact Framework wurde auf die besonderen Bedürfnisse der Windows Phone-Plattform geschnitten. Anstatt die alte Windows Forms-Technologie einzusetzen und diese grafisch aufzupolieren, entschied man sich für neue Wege für die Benutzeroberflächen. Die neue Plattform setzt jetzt vollständig auf Silverlight und auf das XNA Framework. Silverlight findet dabei für die allgemeine Anwendungsentwicklung Verwendung, während das XNA Framework eher für anspruchsvolle 2D- und 3D-Spiele geeignet ist. Ein Teil von Silverlight ist eine auf XAML basierende Oberflächenbeschreibungssprache, die ereignisgesteuerte Einsprungspunkte für den Anwendungsentwickler bietet.

Ein großer Vorteil der Verwendung von Silverlight ist die mögliche Trennung zwischen Oberflächendesign und der eigentlichen Logik der Applikation schon während der Entwicklungszeit. So kann sich ein Grafikdesigner mit der Gestaltung der Oberfläche beschäftigen, während der Entwickler den Code fertigstellt. Genau diese Aufgabenteilung im Blick wurden auch die Entwicklungsumgebungen und Werkzeuge gestaltet.

Patrick Getzmann, Simon Hackfort, Peter Nowak: Entwickeln für Windows Phone 7.5.
Architektur, Frameworks, APIs. Microsoft Press 2011 (ISBN 978-3-86645-465-1)

Dabei bietet Silverlight nicht bloß einen modernen Ersatz für bisherige Oberflächen im Win32- oder Windows Forms-Stil, sondern verbindet grafisch anspruchsvolle Oberflächen und hohe Performance zur Laufzeit mit entwickler- und designerfreundlichen Prozessen schon zur Planungs- und Entstehungsphase der Anwendung.

Das XNA Framework bietet diese Trennung ebenfalls im weitesten Sinne, jedoch ohne separate Entwicklungs- und Designwerkzeuge anzubieten. Es ist letztendlich eine reine Entwicklungsumgebung, die APIs und Funktionen für 2D- und 3D-Spiele oder -Anwendungen bereitstellt. Allerdings ist das XNA Framework keine umfassende 2D- oder 3D-Engine, mit der der nächste Egoshooter oder ähnliches entwickelt werden kann. Es kann aber als Basis für eine solche Engine herangezogen werden. Eine umfangreiche Engine wird aber auch nicht immer benötigt. Einfache 2D-Spiele können auch so ohne eine vollständige Engine in XNA entwickelt werden.

Die bezüglich Silverlight angesprochene Aufgabenteilung läuft hier darauf hinaus, dass der Entwickler, mit oder ohne Hilfe einer Engine, den vollständigen Code und alles, was dazugehört, verantwortet und der Designer benötigte Texturen liefert. Auch hier kann der Code schon vorab fertiggestellt werden. Denn während der Codeentwicklung braucht ja nicht zwangsläufig ein perfekt gerenderter Charakter dargestellt zu werden – ein schlichter Kreis oder ähnliches reicht schon vollkommen aus.

Für welches Framework Sie sich entscheiden, hängt immer ganz von den Anforderungen und der Art der Anwendung ab. Wenn Sie ein Spiel entwickeln wollen, das auch auf dem PC und/oder der Microsoft Xbox 360-Spielekonsole laufen soll, ist XNA das Framework der Wahl. Handelt es sich jedoch um eine Zweckanwendung mit ein paar Steuerelementen wie Schaltflächen, Kontrollkästchen und Listen, wäre Silverlight eindeutig die bessere Wahl. Nicht dass dies in XNA grundsätzlich nicht möglich wäre, aber dort gibt es das Konzept der Steuerelemente nicht, sodass Sie alles nachbauen müssten. Außerdem würde sich eine solche Applikation vom Design her vermutlich stark von anderen Applikationen abheben, und auch die Art und Weise, wie der Anwender die Applikation starten müsste, würde sich komplett von Silverlight-Applikationen unterscheiden.

Das App-Modell in Windows Phone

Das App-Modell in Windows Phone ist weitaus mehr als eine reine Beschreibung der Art und Weise, wie sich Applikationen auf dem Gerät verhalten.

Das App-Modell umfasst die komplette Strategie einer Applikation, angefangen beim Verteilungsprozess bis hin zu den Sicherheitsrichtlinien, die für eine laufende Applikation gelten.

Zuallererst beschreibt es die Form beziehungsweise das Format, in dem eine Applikation ausgeliefert wird. Das App-Modell gibt an dieser Stelle das XAP-Dateiformat vor, das Ihnen schon aus der Silverlight-Entwicklung bekannt sein sollte.

Eine Applikation besteht im Normalfall aus einer Applikations-DLL, Grafiken wie dem SplashScreen oder dem Applikations-Icon, Manifestdateien sowie weiteren Assemblys.

Diese Dateien werden zur Installation in eine so genannte XAP-Datei gepackt. Deren Format entspricht einer herkömmlichen ZIP-Datei, die auch mit kompatiblen Packer-Tools entpackt beziehungsweise selbst erstellt werden kann.

Patrick Getzmann, Simon Hackfort, Peter Nowak: Entwickeln für Windows Phone 7.5.
Architektur, Frameworks, APIs. Microsoft Press 2011 (ISBN 978-3-86645-465-1)

Der eigentliche Auslieferungsprozess ist von der formalen Beschreibung her ziemlich einfach gehalten. Die Applikation wird im Windows Phone Marketplace eingereicht, dort untersucht und anhand der Richtlinien getestet. Nach der Zertifizierung wird die XAP-Datei digital signiert und für den Verkauf freigegeben. Danach kann die Applikation von jedem Anwender je nach Lizenzmodell kostenfrei oder gegen eine Gebühr erworben und somit installiert und genutzt werden.

Das im Rahmen dieses Prozesses gewisse Vor- und, im schlimmsten Fall, Nachbereitungen erforderlich sind, und dementsprechend einige Anforderungen existieren, ist logisch. Diese werden aber ausführlich im Kapitel 11 behandelt.

Das erwähnte Lizenzmodell ist ein weiterer Bestandteil des App-Modells. Da Anwender Applikationen ausschließlich über den Windows Phone Marketplace erwerben können, liegt auch die Lizenzierung komplett in den Händen des Marketplace. Als Entwickler brauchen Sie sich nicht um Dinge wie Registrierung und Lizenzverwaltung zu kümmern – Sie können sich voll und ganz auf die Entwicklung der Funktionalität Ihrer Anwendung konzentrieren.

Der Windows Phone Marketplace stellt mittels eines kryptografischen Verfahrens sicher, dass nur Applikationen ausgeführt und geladen werden können, die auch dort erworben worden sind, ungeachtet dessen, ob die Applikation kostenlos oder kostenpflichtig erhältlich ist.

Einzige Ausnahme bilden so genannte *TrialMode*-Applikationen, also Testversionen, in denen nur ein eingeschränkter Funktionsumfang zur Verfügung steht, oder deren Laufzeit zeitlich begrenzt ist. Hier haben Sie als Applikationsentwickler die Freiheit selbst zu entscheiden, welche Funktionen unterstützt werden sollen, und welche nicht. Dazu müssen Sie allerdings auch entsprechenden Code einbauen, der dies berücksichtigt. Ob Ihre Anwendung aber überhaupt einen TrialMode anbietet, entscheiden Sie selber beim Einreichen der Applikation. Dieser TrialMode wird in Kapitel 11 näher betrachtet.

Schließlich wird im App-Modell noch das Laufzeitverhalten einer Applikation beschrieben. Wie bei der kurzen Beschreibung des XAP-Formats schon erwähnt, besteht eine Anwendung aus einer DLL, Grafiken oder sonstigen Ressourcen sowie zusätzlichen Assemblys und Manifestdateien. Was jedoch fehlt, ist eine ausführbare EXE-Datei. Dies liegt daran, dass die Anwendung nicht als eigenständige Applikation läuft, sondern vom Windows Phone über einen Host-Prozess geladen und ausgeführt wird.

Der Hostprozess sorgt dafür, dass die Applikation in ihrer eigenen Sandbox läuft, also in einer abgesicherten Umgebung. Dabei werden alle System- und Dateizugriffe vom Host-Prozess abgesichert. Das bedeutet, dass Applikationen jeweils nur auf eigene Daten zugreifen können und somit vor Manipulationen geschützt sind. Gleichzeitig wird dadurch auch verhindert, dass die Applikation andere Daten manipulieren und somit fremde Applikationen oder das ganze System gefährden können.

Zusätzlich sind über die abgesicherten Systemzugriffe Daten des Benutzers vor unerwünschten Zugriffen geschützt, da diese nicht mehr direkt von der Applikation durchgeführt werden, sondern lediglich eine Anfrage an das System gestellt wird. So können von einer Applikation nicht mehr automatisch Kontakte ausgelesen oder E-Mails und Kurznachrichten versendet werden, ohne dass der Benutzer dies bemerkt und seine Zustimmung erteilt hat.

Ein weiterer großer Vorteil dieser Separation von Applikationen und Host-Modell ist somit auch das vom System kontrollierte Ausführen von Applikation. Das System hat stets die vollständige Kontrolle über die Verwendung und Zuteilung von Ressourcen wie Speicher, Sensoren oder vor allem auch CPU-Nutzung.

Damit ist gewährleistet, dass der Anwender jederzeit die volle Kontrolle über das Gerät hat, es jederzeit auf Eingaben wie das Drücken der Starttaste reagiert und bedienbar bleibt.

Patrick Getzmann, Simon Hackfort, Peter Nowak: Entwickeln für Windows Phone 7.5.
Architektur, Frameworks, APIs. Microsoft Press 2011 (ISBN 978-3-86645-465-1)

Das UI-Modell in Windows Phone

Das UI-Modell in Windows Phone besteht im Wesentlichen aus zwei Teilen. Zum einem beschreibt das UI-Modell das grundsätzliche Navigationsmodell des Geräts. Dieses Navigationsmodell spiegelt sich auch in Ihrer eigenen Applikation wieder, sofern Sie eine Silverlight-Applikation erstellen. In XNA-Applikationen kommt das Navigationsmodell nicht automatisch zum Tragen, Sie können es allerdings dort selber nachimplementieren, um beispielsweise im Optionsmenü Ihres Spiels navigieren zu können.

Der zweite Bestandteil des UI-Modells ist das *Shell Frame Composing* mit allen seinen Bestandteilen wie der Verwaltung der aktuellen Anzeige, von Seitenübergangseffekten und des zentralen Z-Order Managements.

Das Navigationsmodell

Das Navigationsmodell in Windows Phone ähnelt stark der Navigation von Webseiten in einem Browser. Sämtliche aufgerufenen Applikationen und Dialogfelder werden vom Navigationsmodell in einer so genannten Session abgespeichert. Der Begriff *Dialogfeld* ist hier allerdings nicht wirklich zutreffend, da man eher von einer Page oder Seite sprechen sollte.

Das in Windows Phone verwendete Navigationsmodell ist stark an das Frame- und Page-System in Silverlight angelehnt und ist für Windows Phone adaptiert worden.

Eine Silverlight-Applikation für Windows Phone besteht aus einem Rahmen, dem so genannten Frame, und den eigentlichen Inhalten, also den Seiten oder Pages. Innerhalb des Rahmens haben Sie die Möglichkeit, mehrere Seiten anzuzeigen, wobei immer nur eine Seite aktiv ist und angezeigt werden kann. Die anderen Seiten können über eigenen Code angesteuert werden. Ausnahme ist hier die Zurück-Taste, die wie der Zurück-Button eines Webbrowsers funktioniert.

Das Navigationsmodell speichert aber nicht nur die Page-Reihenfolge Ihrer Anwendung, sondern auch die aller anderen Anwendungen, die innerhalb einer Session aufgerufen wurden, sofern es sich nicht um XNA-Spiele handelt. Innerhalb eines XNA-Spiels gibt es dagegen keine derartigen Pages. Daher ist dort auch keine automatische Navigation möglich.

Wichtig ist an dieser Stelle noch, dass innerhalb einer Session jede Anwendung nur einmal vorkommt. Wird eine Anwendung erneut gestartet, werden vorherige Navigationseinträge aus der Session entfernt.

Damit sich für den Anwender ein gleich bleibendes Verhalten ergibt, ist das Manipulieren der Session-Informationen nicht möglich. Auch ein direktes Beenden einer Silverlight-Applikation ist aus dem gleichen Grund nicht implementiert. Je nach Art und Weise der Navigation wird die Applikation automatisch beendet oder nur in einen Pausenzustand versetzt. Weitere Details dazu finden Sie in Kapitel 6.

Shell Frame Management

Das Shell Frame Management ist ein wichtiger Bestandteil des UI-Modells. Es ist verantwortlich für das Composing – das Zusammensetzen der Gesamtansicht der Windows Phone-Oberfläche.

Dieses Composing gewährleistet, dass Bildschirmbereiche wie beispielsweise für Telefongespräche oder die Tastatureingabe jederzeit angezeigt werden können und nicht von Applikationen überdeckt werden.

Der prinzipielle Aufbau des Bildschirms gestaltet sich wie folgt:

Patrick Getzmann, Simon Hackfort, Peter Nowak: Entwickeln für Windows Phone 7.5.
Architektur, Frameworks, APIs. Microsoft Press 2011 (ISBN 978-3-86645-465-1)

Abbildung 3.2 Verschiedene Oberflächenbeispiele

Die unterste Schicht der Ansicht bildet das *Direct3D Surface*, über das die nachfolgenden Grafiken gelegt werden. Diese Oberfläche wird über die Direct3D-Schnittstelle direkt auf den Bildschirm gezeichnet. Das hat den großen Vorteil, dass alle Fenster über schnelle Operationen im Speicher gezeichnet werden können und jeweils nur eine einzelne Zeichenoperation auf der eigentlichen Hardware durchgeführt wird.

Das Start-Bild von Windows Phone sowie der Bildschirmbereich von Applikationen werden zuerst gezeichnet. Innerhalb einer Applikation können zwei weitere Systemfenster verwendet und angezeigt werden. Zum einen kann die Softwaretastatur zur Texteingabe eingeblendet werden, zum anderen kann eine Applikation eine *App Bar* verwenden, die am unteren Bildschirmbereich angezeigt wird.

Für den Fall eines eingehenden Telefonanrufs wird das Fenster der Telefonapplikation an dieser Stelle über alle vorherigen Fenster gezeichnet. Dazu erhält dieses Fenster den Fokus.

Zu guter Letzt gibt es noch drei weitere Fenster, die angezeigt werden können. Dies sind die Statusleiste, das Benachrichtigungsfenster (Toast) und die Lautstärkeanzeige.

Dieses globale Zusammensetzen der Ansicht durch das System gewährleistet ein konsistentes Verhalten der Geräteoberfläche. Weil Applikationsfenster nicht den gesamten Bildschirm exklusiv für sich beanspruchen können, ist aber auch eine erhöhte Bedienbarkeit für den Anwender sichergestellt, da etwa eingehende Telefonanrufe nicht durch Applikationen blockiert werden können.

Neben dem reinen Zeichnen der Oberfläche ist das Shell Frame Management für den Wechsel zwischen den verschiedenen Applikationen oder Seiten zuständig und kann globale Übergangsanimationen oder sonstige Effekte steuern.

Cloud Services

Windows Phone ist stark auf Cloud-Computing ausgelegt und hat schon einige Web-Dienste von Hause aus integriert.

Im Vordergrund steht der Windows Live-Dienst, über den sich der Anwender am Gerät authentifiziert. Ist an die Live ID auch schon ein Windows Live Hotmail-Konto gebunden, steht der Synchronisation von E-Mails, Kalender und Kontakten nichts mehr im Weg. Zusätzlich zu Hotmail ist der Xbox Live-Dienst in Windows Phone integriert.

Patrick Getzmann, Simon Hackfort, Peter Nowak: Entwickeln für Windows Phone 7.5.
Architektur, Frameworks, APIs. Microsoft Press 2011 (ISBN 978-3-86645-465-1)

Es gibt einen eigenen *Game-Hub*, in dem der Xbox Live-Avatar sowie Erfolge angezeigt werden. Einige Spiele werden eine Xbox Live-Unterstützung enthalten, sodass von Windows Phone aus Spielerfolge erzielt werden können und rundenbasierte Spiele auch im Austausch mit einer Xbox 360 möglich sind.

Dem Bing-Dienst wird sogar eine eigene standardisierte Hardwaretaste zugeteilt, um jederzeit ein schnelles und einfaches Suchen zu ermöglichen. Die Suche nach QR-Codes, das Übersetzen von Texten und das Erkennen von Liedern ist jetzt mit Bing möglich.

Hinzu kommen noch zwei für Windows Phone spezifische Dienste.

Der Windows Phone Location-Dienst ermöglicht die Abfrage von geobasierten Informationen und bietet somit die Möglichkeit, standortbasierte Funktionen in eigene Applikationen zu integrieren.

Der zweite dieser Dienste ist der Push Notification Service. Über diesen Dienst ist es möglich, einem eigenen Webdienst eine URL mitzuteilen, über die die Applikation erreichbar ist, auch wenn die Applikation schon beendet wurde. Gibt es innerhalb des Webdienstes eine neue Information, die die Anwendung erhalten soll, wendet sich der Webdienst an die URL. Hinter dieser URL steckt der Push Notification Service, der dann eine Benachrichtigung an das Gerät sendet und dem Anwender die Möglichkeit gibt, anhand der kurzen Information die Applikation zu starten.

Ein weiterer fest integrierter Bestandteil ist das soziale Netzwerk Facebook. Sofern ein Facebook-Account konfiguriert ist, werden automatisch die Kontakte aus Facebook geladen und angezeigt. Neuigkeiten über einzelne Personen können angezeigt werden. Der eigene Status kann natürlich ebenfalls schnell und einfach aktualisiert werden. Über die Foto-Applikation können neu aufgenommene Fotos direkt auf Facebook bereitgestellt werden. Seit Windows Phone 7.5 sind Twitter und LinkedIn ebenfalls in die Plattform integriert.

Natürlich bietet Windows Phone über seine APIs auch die Möglichkeit, andere Webdienste anzusprechen, egal ob diese bereits existieren oder ob sie von Ihnen selbst erstellt werden.

Der Windows Phone-Kernel

In der Beschreibung der Anwendungsschicht wurde erwähnt, dass Windows Phone ein auf .NET-Technologie basierendes System ist. Dies ist aber nicht ganz korrekt formuliert gewesen, denn tief unter der Haube steckt der eigenständige Windows Phone-Kernel. Dieser Kernel bildet die softwaretechnische Basis des neuen Betriebssystems und umfasst Treiber für die Hardware, den Netzwerkstack, die Dateizugriffsteuerung sowie das komplette Sicherheitsmodell.

Die Basiskomponente des Kernels ist, wie bei vielen anderen Systemen, ein Windows Embedded-Betriebssystem. Die dort enthaltenen Treiber und DLLs wie beispielsweise der Netzwerkstack oder Audio-/Video-Codecs liegen dort größtenteils als native C/C++-DLLs vor. Der Kernel ist allerdings nur für den jeweiligen Hardwarehersteller zugänglich, und selbst dieser unterliegt, wegen der Spezifikationen seitens Microsoft, diversen Einschränkungen, um die Stabilität und Performance der Plattform nicht zu gefährden.

Details über den Aufbau des Kernels, der Treiber, sowie den zur Verfügung stehenden erweiterten nativen und managed APIs sind nicht öffentlich zugänglich, sodass Sie als normaler Anwendungsentwickler damit auch nicht in Berührung kommen werden.

Patrick Getzmann, Simon Hackfort, Peter Nowak: Entwickeln für Windows Phone 7.5.
Architektur, Frameworks, APIs. Microsoft Press 2011 (ISBN 978-3-86645-465-1)

Das Metro-Design

In Verbindung mit Windows Phone spricht man gleichzeitig vom *Metro-Design*. Dieses Design wurde speziell für diese Plattform entwickelt und auf die gesamte Benutzeroberfläche von Windows Phone angewendet.

Pate für dieses Design standen, wie der Name schon vermuten lässt, Bahnhöfe und Flughäfen. Damit ist die Visualisierung von Informationen auf Bahnhöfen und Flughäfen in Form der Beschilderung gemeint. Hauptziel ist es, die Informationen so klar und einfach wie möglich darzustellen. Die Information selbst ist daher im Mittelpunkt und an die Visualisierung angepasst. Dabei wird die Information entweder durch Piktogramme, durch gut lesbaren Text oder in einer Kombination aus beidem transportiert. Ziel ist es, dem Benutzer so einfach und so schnell wie möglich die Informationen bereit zu stellen, was zwangsläufig zu einem eher schlichten Design führt.

Abbildung 3.3 »Paten« für das Metro-Design (Quelle: UI Design and Interaction Guide for Windows Phone 7)

Der Anwender hingegen empfindet dieses Design durchaus als »schick«, obwohl es minimalistisch anmutet. Schließlich ist das Design für den Anwender nichts Neues, da er es bereits aus dem täglichen Leben kennt. Die Darstellungsweise kommt dem Anwender zugute, weil die Information im Vordergrund steht. Dies ist ein Ziel des Metro-Designs.

Andere Designs propagieren eher einen so genannten *Glossy*-Stil. Dieser ist meist bunt, verwendet viele Farbverläufe, nutzt eine leichte dreidimensionale Präsentation der genutzten Steuerelemente und wirkt oft sehr verspielt. Dies führt dazu, dass die Präsentationsart der zu visualisierenden Daten ebenfalls als angenehm oder interessant empfunden wird, doch rückt so der Inhalt leicht ein wenig in den Hintergrund, was das Lesen erschwert.

Ein weiterer Eckpfeiler von Metro ist die Darstellung von Text, wenn die Darstellung durch Piktogramme nicht sinnvoll ist. So ist die zur Visualisierung verwendete Schriftart essentiell. Schlichte Schriftarten lassen sich leicht und schnell lesen, während geschnörkelte Schriftarten die Informationen eher schlecht zum Anwender transportieren. Dazu kommt, dass die Informationen auch prägnant auf den Punkt gebracht werden sollen und im direkten Kontext zum Anwender stehen. Wichtige Informationen sollen zur leichteren Erkennung in einer größeren Schriftgröße dargestellt werden, zusätzliche Sekundärinformationen in einer kleineren.

Patrick Getzmann, Simon Hackfort, Peter Nowak: Entwickeln für Windows Phone 7.5.
Architektur, Frameworks, APIs. Microsoft Press 2011 (ISBN 978-3-86645-465-1)

Abbildung 3.4 Weitere Informationskacheln aus dem alltäglichen Leben

Ein Beispiel für die Symbolik von Metro ist, wenn man anstatt einer Schaltfläche zur Wiedergabe von Musik mit dem Text *Play* ein Dreieck zum Abspielen nutzt, wie es auf den meisten Abspielgeräten bereits verwendet wird.

Diese Designprinzipien der Darstellungsart zeigen bereits die Grundidee hinter Metro auf: Der Anwender soll immer im Vordergrund stehen. Er soll schnell auf *seine persönlichen* Informationen zugreifen und diese verarbeiten können. Dies führt jedoch automatisch zu einigen weiteren Anforderungen:

- Der Anwender soll eher *intuitiv* und *persönlich* entscheiden, wie er mit einer Anwendung arbeitet, und nicht durch eine Anwendung »gegängelt« werden. Eine Arbeitsweise soll daher nicht in ein Schema gepresst werden.
- Es sollen nach Möglichkeit nur relevante, vielleicht auch nur ortsbezogene Daten dargestellt werden. Der Anwender soll seine Daten optimalerweise nicht suchen müssen. Darstellungen sollen präzise sein ohne abzulenken.
- Informationen sollen nach Möglichkeit verbunden sein. So geht es immer um die Informationen eines Benutzers, die für ihn natürlich zusammengeführt zur Verfügung stehen. Es ist nicht in seinem Sinne, Informationen aus verschiedenen Quellen zusammensuchen zu müssen.

Mindestens eine dieser Anforderungen sollte von jeder zu erstellenden Anwendung beachtet werden.

Neben der visuellen Darstellung von Informationen ist im Metro-Design auch die Präsentation der Informationen wichtig. Einfach Inhalte nur statisch darzustellen würde zwar zum generellen Modell der »Schlichtheit« passen, doch ist dies eher kontraproduktiv. So ist es eher sinnvoll, ein wenig »Bewegung« in die Anwendung zu bringen.

Was im ersten Augenblick wie ein Widerspruch klingt, ergibt bei näherer Betrachtung einen tieferen Sinn. Die hinter Metro steckende Idee, Informationen auf eine Art und Weise zu präsentieren, die auf Bahnhöfen und Flughäfen genutzt wird, stellt bereits die Verbindung zur physischen Welt her. Genauso verhält es sich bei der Präsentationsart. Es ist eine eher physische Anmutung, dass das Schließen einer Tür wie eine »Animation« wirkt. Sie ändert nicht einfach schlagartig ihren Zustand von *geöffnet* zu *geschlossen*. Vielmehr schließt sie sich gleitend über eine Zeitspanne hinweg. Dies mag an dieser Stelle zwar sehr abstrakt klingen, doch spiegelt es die Realität wider.

Patrick Getzmann, Simon Hackfort, Peter Nowak: Entwickeln für Windows Phone 7.5.
Architektur, Frameworks, APIs. Microsoft Press 2011 (ISBN 978-3-86645-465-1)

Verinnerlicht man dieses abstrakte Beispiel, so merkt man, dass es eher anwendbar auf die Realität ist, als ein reiner Statuswechsel. Angenommen, es fällt einem ein Glas aus der Hand. So ist zwischen *wird gehalten* und *liegt auf dem Boden* noch der Zustand *fällt zu Boden* eingefügt. Es lässt sich davon ableiten, dass ein Statuswechsel streng genommen immer einen Zwischenstatus besitzt, der sich mit einer »Animation« verknüpfen lässt.

So ist ein weiteres Ziel des Metro-Designs die Nutzung von Animationen. Dabei gilt hier jedoch erneut eine Einschränkung: Animationen sollen nach Möglichkeit *dezent* genutzt werden. Nicht alles, was sich bewegen und animieren lässt, soll auch animiert werden. Damit wird der Anwender eher überfordert. Man erinnere sich nur an die frühen Zeiten von Visual Basic 1.0 im Jahre 1991. Ab dieser Version war es möglich, Hintergrundfarben in Steuerelementen zu verwenden und dies wurde auch bis zum Exzess getan, einfach nur, weil es nun mal möglich war. Dies ist jedoch nicht das Ziel. Eher geht es um ein sinnvolles und gezieltes Einsetzen von Animationen. Eine Liste, die sich öffnet, sollte sich animiert öffnen. Das Wechseln von Oberflächen kann auch animiert werden. Dadurch wirkt die Darstellung lebendig und nicht statisch.

Zur Animation gesellt sich noch die Bewegung. Werden graphische (Steuer-)Elemente wortwörtlich angefasst, so erwartet der Benutzer auch, dass diese sich mitbewegen. Eine Liste beispielsweise sollte sich daher auch bewegen, wenn der Anwender diese nach oben oder unten verschiebt. »Schubst« dieser hingegen eine Liste an, indem er das grafische Element während einer Bewegung verlässt, so sollte diese sich auch fortbewegen und sich während dieser animierten Darstellung verlangsamen, bis sie letztendlich zum Stillstand kommt. Dem Anwender macht hierdurch die Verwendung der Anwendung *Spaß*, egal ob es sich nun um Arbeit oder privates Vergnügen handelt. Dabei ist es immens wichtig, dass diese Animationen natürlich wirken und nicht »ruckeln«.

Es ist diese *Einfachheit* und *Intuition*, auf die es ankommt. Der Anwender möchte keine Gebrauchsanweisung vorab lesen müssen, um eine Anwendung zu bedienen. Optimal ist es, wenn sich die Anwendung selbst erklärt und somit einfach und direkt bedienbar ist. So ist es aus reiner Anwendersicht nur logisch, dass beispielsweise bei der Darstellung eines Bildes das Bild vergrößert oder gedreht werden kann. Dies wird in der Regel mit mehreren Fingern auf dem Display bzw. durch eine Rotationsbewegung des Geräts durchgeführt.

Bei der Eingabe von Daten mittels Tastatur sollte der Benutzer genauso unterstützt werden. Diesem immer eine vollwertige Tastatur darzustellen ist nicht sinnvoll, da möglicherweise Elemente dargestellt werden, die nicht benötigt werden. Ein angepasstes Tastaturlayout, welches die Eingabe einer Telefonnummer oder einer SMS erleichtert, ist hier der sinnvollere Weg. Somit werden Fehleingaben beispielsweise ausgeschlossen, wenn die Eingabe von Buchstaben bei einer Telefonnummer verhindert wird.

Die Anwendungsoberfläche muss auch nicht auf den optischen Bereich beschränkt werden. So kann es sinnvoll sein, über den sichtbaren Bereich hinaus Informationen darzustellen, zu denen der Benutzer navigieren kann.

Werden Anwendungseinstellungen angeboten, so sollten diese nicht noch zusätzlich mittels *OK* oder *Übernehmen* oder ähnlichem akzeptiert werden müssen. Der Anwender hat seine persönlichen Einstellungen ja bereits vorgenommen, sodass eine Bestätigung dieser, streng genommen, überflüssig ist.

All diese Punkte sind im Leitsatz von Metro zu finden:

- *METRO IST UNSERE DESIGNSPRACHE. WIR NENNEN SIE METRO, WEIL SIE MODERN UND KLAR IST. SIE IST SCHNELL UND LEBENDIG. ES GEHT UM INHALT UND TYPOGRAPHIE. UND SIE IST VOLLKOMMEN AUTHENTISCH*

  (Quelle: *Windows Phone Design System – Metro*)

Es gibt darüber hinaus jedoch noch weitere Punkte, die nicht direkt die Darstellungsart betreffen. So erwartet ein Anwender beim physischen Drehen des Geräts, dass auch die Darstellung angepasst wird, oder anders gesagt, sich »mitdreht«. So wird das Hochformat, auch Portrait genannt, auf das Querformat, auch Landscape genannt, umgestellt und umgekehrt.

All diese Szenarien werden durch die in Windows Phone eingesetzte Hardware ermöglicht und hinterlassen somit einen positiven Eindruck beim Anwender.

An dieser Stelle fällt bereits auf, dass das Metro-Design nur ein Teilbereich des Metro-Systems ist. Metro allein beschreibt nicht nur, wie Informationen visualisiert werden sollen, sondern wie die Interaktion mit dem Benutzer darüber hinaus aussehen sollte. Hierzu gehört die Integration der Anwendung, sollte Metro genutzt werden, in das Gerät. So kann der Anwender das Farbschema von Metro seinem Geschmack nach anpassen, was sich in den Anwendungen auch widerspiegelt. Dies sollte jedoch auch von der eigenen Anwendung unterstützt werden, wodurch gegenüber dem Anwender das Gefühl entsteht, dass die Anwendungen zusammengenommen »aus einem Guss« sind.

Das Interessante am Metro-System ist jedoch, dass es ausdrücklich darauf hinweist, dass der Einsatz des Metro-Designs nicht immer sinnvoll ist. Es soll daher auch nicht immer verwendet werden!

Auch dies klingt im ersten Augenblick eher verwunderlich, doch ergibt es durchaus Sinn. So beschreibt es doch, dass die Natürlichkeit hin zum Anwender gegeben sein soll. Wird nun beispielsweise eine Anwendung erstellt, die ein Piano darstellt, so sollte diese das auch tun. Es ist selbsterklärend, wenn der Anwender eine Klaviatur sieht, dass es sich um ein Piano handelt. Er weiß, wie er es zu verwenden hat. Ein anderes Beispiel ist ein persönliches Tagebuch. Dies sollte auch wie ein Buch aussehen und sich so »anfühlen«. Bei diesen Beispielen wäre der Einsatz des Metro-Designs eher hinderlich.

Metro bietet somit ein interessantes System an. Es kann die Erfahrung eines Anwenders positiv beeinflussen und sollte somit immer in Betracht bezogen werden, wo der Einsatz sinnvoll ist.

Mittlerweile wird Metro im Dashboard der Xbox 360 eingesetzt und auch Windows 8 wird eine Metro-Style UI besitzen.

Mehr zum Thema Metro erfahren Sie in den englischsprachigen Dokumenten *Windows Phone Design System – Codename Metro* und *UI Design and Interaction Guide for Windows Phone 7*, die im Internet unter *http://create.msdn.com* frei erhältlich sind.

TIPP Es ist hinlänglich bekannt, dass Entwickler nur selten gute Designer sind. Entwickler sind eher funktional getrieben und die Optik spielt beim Entwickeln eine untergeordnete Rolle.

Gibt es nun keinen Designer, den man in der Entwicklung zu Rate ziehen kann, kann man sich jedoch anderweitig behelfen. Suchen Sie sich jemanden aus Ihrem näheren Umfeld, der am besten mit Softwareentwicklung und dergleichen wenig zu tun hat und auch keine Scheu besitzt, die Wahrheit zu sagen.

Lassen Sie diesen mit nicht mehr an Informationen als unbedingt nötig einen Prototyp der zu entwickelnden Anwendung testen. Sie erhalten so wichtige Informationen, wie die Anwendung auf einen Benutzer wirkt.

Patrick Getzmann, Simon Hackfort, Peter Nowak: Entwickeln für Windows Phone 7.5.
Architektur, Frameworks, APIs. Microsoft Press 2011 (ISBN 978-3-86645-465-1)

Entwicklung mit dem Windows Phone

Der Großteil der in den nachfolgenden Kapiteln beschrieben Technologien, APIs und Beispiele lässt sich mit dem Windows Phone-Emulator durchführen. Einige wenige Funktionen jedoch können nicht emuliert werden und müssen auf einem Windows Phone-Gerät durchgeführt werden.

Daher möchten wir an dieser Stelle auf Kapitel 11 verweisen, in dem die Prozesse beschrieben sind, wie Sie Ihr Windows Phone als Entwicklergerät aktivieren können, sowie die grundlegenden Schritte zur Verwendung der Zune-Desktop-Software und des Windows Phone Connect-Dienstprogramms.

NuGet

NuGet ist ein populärer Paketmanager für das Visual Studio. Er ermöglicht es, Entwicklungsbibliotheken zu installieren und zu konfigurieren, wie auch zu aktualisieren.

Zwar klingt dies an sich nicht spannend, doch erleichtert es die Entwicklung ungemein. Angenommen es soll eine Bibliothek verwendet werden, welche eine Referenz auf eine andere Bibliothek besitzt, die auch installiert werden müsste, so ist nicht nur auf die richtige Version der referenzierten Bibliothek zu achten, sondern die Bibliotheken müssen auch noch in das Projekt integriert werden sowie gegebenenfalls noch Dateipfade angepasst werden.

Um all diese Probleme kann sich NuGet kümmern und ist dabei einfach aus dem Visual Studio heraus bedienbar. Da einige der im Buch vorgestellten Bibliotheken auch über NuGet angeboten werden, lohnt ein kurzer Blick auf NuGet.

NuGet als Erweiterung für Visual Studio ist unter *http://nuget.org/* erhältlich und schnell installiert, sodass sich nach der Installation in Visual Studio 2010 unter *Tools* der neue Eintrag *Library Package Manager* findet.

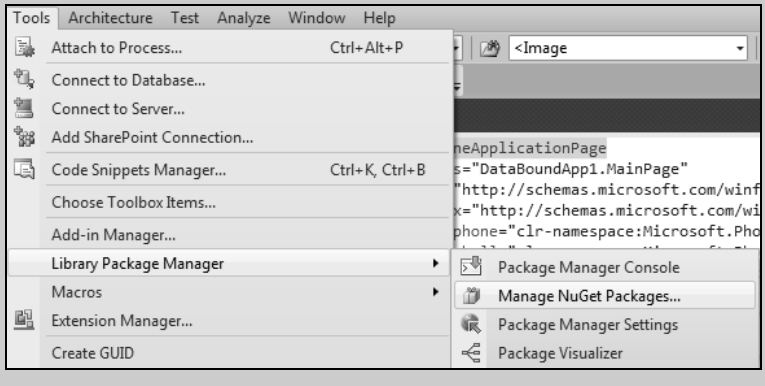

Abbildung 3.5 NuGet, eingebunden in Visual Studio 2010

Aktiviert man den dargestellten Eintrag *Manage NuGet Packages…* so erhält man ein neues Dialogfeld.

Wechselt man dort auf in die Kategorie *Online* kann man beispielsweise im zugehörigen Suchfeld die Zeichenfolge *Silverlight Toolkit* angeben, die dann folgendes Ergebnis darstellt. ▶

Patrick Getzmann, Simon Hackfort, Peter Nowak: Entwickeln für Windows Phone 7.5.
Architektur, Frameworks, APIs. Microsoft Press 2011 (ISBN 978-3-86645-465-1)

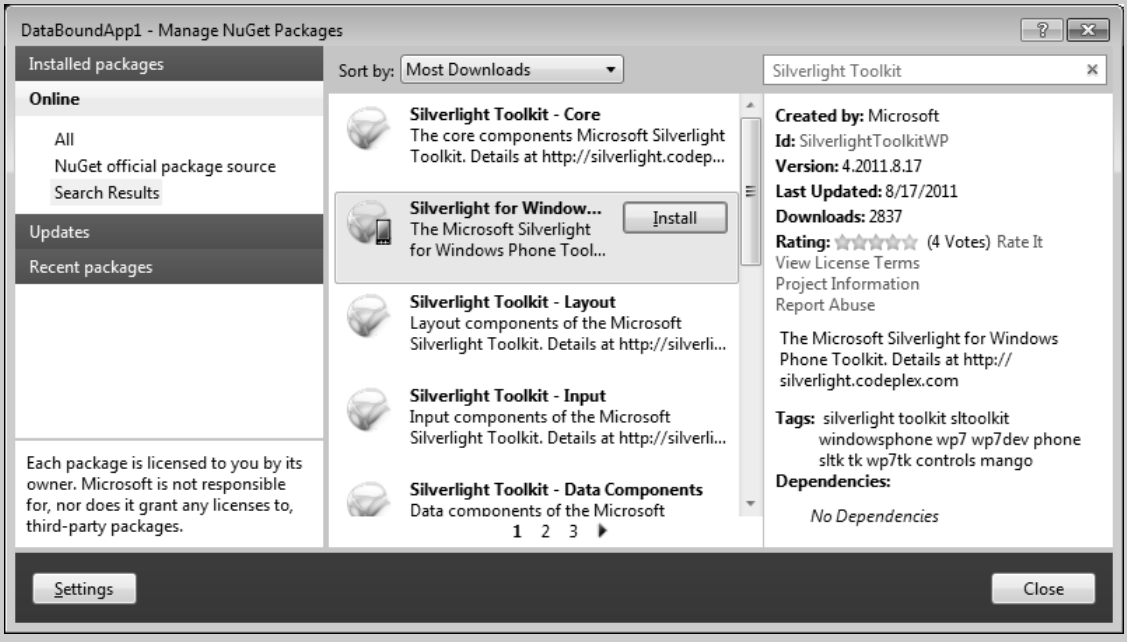

Abbildung 3.6 Das Silverlight Toolkit für Windows Phone via NuGet

Wählt man zum zugehörigen Eintrag die Schaltfläche *Install* aus, so beginnt die Installation und Konfiguration des entsprechenden Pakets.

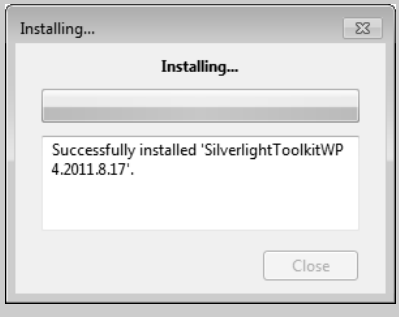

Abbildung 3.7 Installationsdialogfeld zum zugehörigen Paket

Um ein installiertes Paket wieder zu entfernen, wechseln Sie im gleichen Dialogfeld in den Bereich *Installed Packages*. Dort sehen Sie alle in diesem Projekt installierte Bibliotheken. Soll hiervon eine Bibliothek entfernt werden, so genügt die Aktivierung der Schaltfläche *Uninstall* im zugehörigen Eintrag.

Patrick Getzmann, Simon Hackfort, Peter Nowak: Entwickeln für Windows Phone 7.5.
Architektur, Frameworks, APIs. Microsoft Press 2011 (ISBN 978-3-86645-465-1)

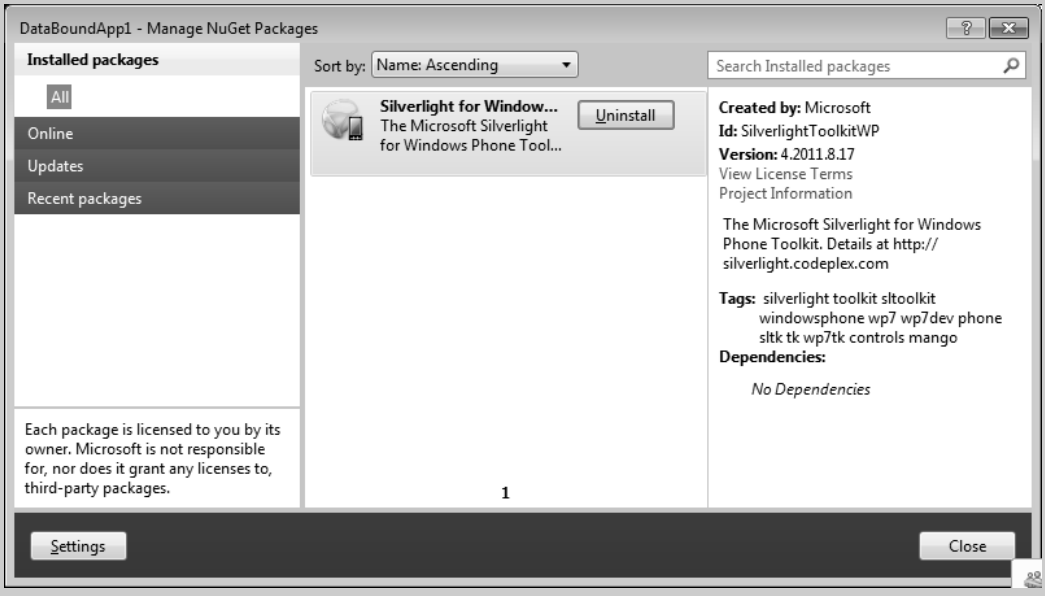

Abbildung 3.8 Ein installiertes Paket ist genauso schnell deinstalliert

Über den Bereich Updates sind Aktualisierungen für eine Bibliothek erhältlich. Einfacher kann das Paketmanagement nicht sein.

Ist die vorher beschriebene Installation erfolgreich durchgeführt worden und keine Deinstallation erfolgt, so ist die zugehörige Bibliothek dann im Bereich *References* der Projekt-Solution eingebunden.

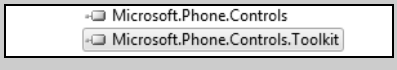

Abbildung 3.9 Die Referenzen sind durch die Installation über NuGet gesetzt

Mehr zu NuGet erfahren Sie unter *http://nuget.org/* und *http://nuget.codeplex.com*.

Patrick Getzmann, Simon Hackfort, Peter Nowak: Entwickeln für Windows Phone 7.5.
Architektur, Frameworks, APIs. Microsoft Press 2011 (ISBN 978-3-86645-465-1)