Tombstoning in Windows Phone 7

Elke WP7 ontwikkelaar moet er rekening mee houden

Maarten Struys

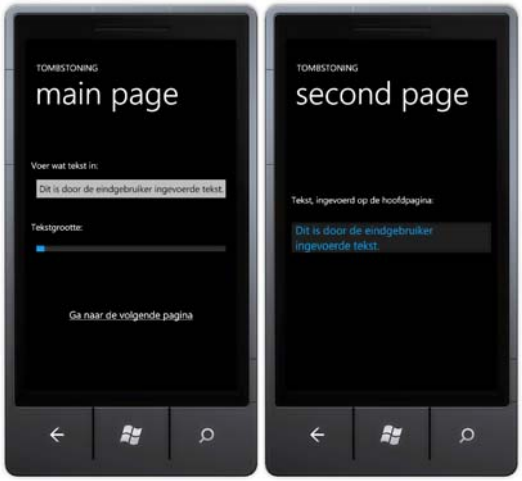
Deze titel lijkt misschien de start van een macaber verhaal, een verhaal over grafstenen in combinatie met Windows Phone 7. Niets is echter minder waar. Het verhaal gaat veel meer over het tot leven wekken van applicaties op de telefoon. Dit artikel beschrijft een fenomeen waar elke Windows Phone 7 softwareontwikkelaar zich van bewust moet zijn.

Een telefoongebruiker navigeert veelvuldig tussen verschillende schermen van verschillende applicaties. Omdat op een willekeurig moment maar één applicatie op de telefoon zichtbaar is, en een gebruiker zonder daarvan bewust te zijn, snel tussen verschillende applicaties wil schakelen, zijn applicaties voor een deel zelf verantwoordelijk voor het bijhouden van hun toestand. Daardoor lijkt het voor de eindgebruiker alsof een applicatie gewoon doorwerkt zodra die weer zichtbaar is, terwijl de applicatie waarschijnlijk helemaal gestopt was.

Niet alleen een gebruiker, maar ook een applicatie zelf kan ervoor zorgen dat deze tijdelijk wordt onderbroken door zogenaamde Launchers en Choosers aan te roepen. In beide gevallen wordt vanuit een bepaalde applicatie een andere applicatie opgestart. Daarnaast is de telefoon zelf ook verantwoordelijk voor onderbreken van applicaties. Bijvoorbeeld bij binnenkomende telefoongesprekken of als de telefoon in slaap valt. Omdat het gedrag in al deze gevallen nagenoeg identiek is aan dat van een navigerende gebruiker, beperken we ons in dit artikel tot gebruikersacties die Tombstoning veroorzaken.

Aan de hand van een voorbeeldapplicatie onderzoeken we allereerst hoe het applicatiemodel van een Windows Phone 7 Silverlight applicatie er uit ziet. De toepassing zelf is heel simpel, het gaat hier niet om de getoonde functionaliteit, maar uitsluitend om de achterliggende gedachtes. Om zelf goede applicaties te kunnen ontwikkelen die door eindgebruikers gewaardeerd worden is het van essentieel belang om Tombstoning te doorgronden en juist op dit fenomeen te reageren.

De voorbeeldapplicatie die centraal staat in dit artikel bevat twee verschillende schermen. Op het hoofdscherm bevindt zich een TextBox, waar een gebruiker tekst in kan voeren. De grootte van de TextBox kan met een Slider aangepast worden. Als de gebruiker naar het volgende scherm van de applicatie navigeert, wordt daar de eerder ingevoerde tekst zichtbaar.



FIGUUR 1: VOORBEELDAPPLICATIE IN ACTIE.

Interessant om te onderzoeken is, wat er gebeurt als de gebruiker (in plaats van naar het volgende scherm te navigeren) op de 'back' toets of op de starttoets van de telefoon drukt vanuit het hoofdscherm. In beide gevallen wordt onze voorbeeldapplicatie onzichtbaar. Echter, afhankelijk van de gekozen toets keert de gebruiker terug naar een eerder actieve applicatie of navigeert juist naar het startscherm van de telefoon. In het eerste geval (gebruikmaken van de 'back' toets) wordt de applicatie afgesloten en laat bij opnieuw starten een initieel beginscherm zien, dus zonder tekst ingevoerd in de TextBox.

Het tweede geval is interessanter. Vanuit het startscherm start een gebruiker normaal gesproken een andere toepassing op de telefoon. Na verloop van tijd kan het echter zo zijn dat de gebruiker via de 'back' toets teruggaat naar eerder actieve applicaties, waaronder uiteindelijk onze voorbeeldapplicatie. Zonder iets extra's in onze applicatie te doen is ook nu weer een initieel hoofdscherm zichtbaar, dus zonder tekst ingevoerd in de tekstbox. De gebruiker van de telefoon verwacht waarschijnlijk dat onze applicatie zichtbaar

wordt in dezelfde toestand die de applicatie had toen de 'back' toets werd ingedrukt. Om dit te realiseren moet de tekst in de TextBox die de gebruiker had ingevoerd weer volkomen identiek zichtbaar zijn en moet ook de Slider op dezelfde positie staan die deze had bij het wegnavigeren van de applicatie. Door te analyseren welke code wordt doorlopen is het gedrag van de applicatie in beeld te krijgen. Dit kunnen we doen door gebruik te maken van toevoegen van debug teksten, die in Visual Studio 2010 worden getoond als de applicatie op de telefoon (emulator) wordt uitgevoerd. De applicatiecode die actief is bij zichtbaar worden en verwijderen van hoofdscherm is te zien in het volgende codefragment.

```
public partial class MainPage : PhoneApplicationPage
{
    public MainPage()
    {
        InitializeComponent();
        Debug.WriteLine("MainPage Constructor");
    }


protected override void OnNavigatedFrom(System.Windows.Navigation.
NavigationEventArgs e)
    {
        base.OnNavigatedFrom(e);

Debug.WriteLine("MainPage OnNavigatedFrom - Uri = " + e.Uri.To
String());
    }


protected override void OnNavigatedTo(System.Windows.Navigation.
NavigationEventArgs e)
    {
        base.OnNavigatedTo(e);
        Debug.WriteLine("MainPage OnNavigatedTo - Uri = " + e.Uri.
ToString());
    }
}
```

LISTING 1: NAVIGEREN NAAR EN VAN HET HOOFDSCHERM.

Laten we eens kijken wat er precies gebeurd op het moment dat de gebruiker de applicatie start en stopt via de backtoets. Vervolgens kijken we ook wat er gebeurt als de gebruiker de applicatie start, vervolgens een nieuwe applicatie start en terugkeert naar onze voorbeeldapplicatie via de backtoets alvorens de applicatie te stoppen via de backtoets.

Bij opstarten van de applicatie wordt de code in de constructor uitgevoerd, vervolgens de code in de OnNavigatedTo method, die telkens wordt aangeroepen als de gebruiker naar het hoofdscherm in onze voorbeeldapplicatie navigeert. Zodra de gebruiker via de backtoets de applicatie verlaat, wordt de code in de OnNavigatedFrom method uitgevoerd. Deze code wordt iedere keer uitgevoerd als de gebruiker het hoofdscherm verlaat, dus niet alleen bij afsluiten van de applicatie, maar ook bij navigeren naar een volgend scherm van de applicatie. Tenslotte is zichtbaar dat het programma wordt beëindigd.

In het geval dat de gebruiker tijdelijk de voorbeeldapplicatie verlaat, lijkt het gedrag in eerste instantie identiek aan het eerder beschreven gedrag bij afsluiten van de applicatie. Echter, nadat de gebruiker via de starttoets terugkeert naar het startscherm op de telefoon sluit de Visual Studio 2010 debugger niet af. De threads van de applicatie worden weliswaar afgesloten, maar het programma zelf niet. Hiermee hebben we het fenomeen Tombstoning in de Visual Studio 2010 debugger zichtbaar gemaakt.

Als de gebruiker via de backtoets terugkeert naar onze applicatie zal opnieuw de constructor worden uitgevoerd en ook de OnNavigatedTo method. Sterker nog, vanuit het hoofdscherm is niet te achterhalen of de applicatie opnieuw wordt gestart of terugkeert vanuit een Tombstone situatie. Doordat de constructor opnieuw wordt aangeroepen is duidelijk dat een nieuwe instantie van het hoofdscherm wordt aangemaakt. We kunnen dus niet zonder meer de situatie reproduceren waarin de applicatie zich bevond toen de gebruiker via de starttoets onze applicatie verliet.

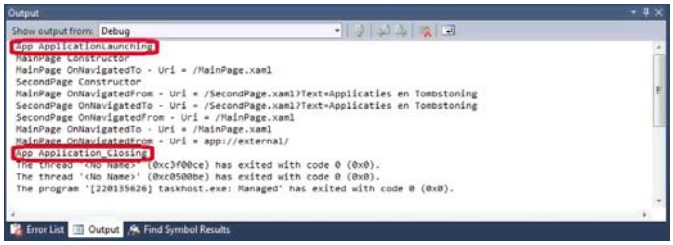
Om dit toch te kunnen moeten een aantal andere gebeurtenissen in de applicatie worden gevolgd. Daarvoor moeten we kijken in de opstartcode die te vinden is in het bestand App.xaml.cs. Deze code wordt bij het maken van een nieuwe applicatie voor Windows Phone 7 door Visual Studio 2010 gegenereerd. In App.xaml.cs vindt applicatie-initialisatie plaats. Ook zijn een aantal lege maar van commentaar voorziene event handlers in deze code te vinden. Deze event handlers worden aangeroepen tijdens starten, opnieuw activeren, afsluiten en verlaten van de applicatie. Door het toevoegen van debug informatie, zoals te zien is in listing 2, kunnen we de eerder beschreven acties herhalen en bekijken of er een verschil is tussen afsluiten van een applicatie en Tombstoning.

```
// Code to execute when the application is launching (eg, from
Start)
        // This code will not execute when the application is
reactivated
        private void Application_Launching(object sender,
LaunchingEventArgs e)
        {
            Debug.WriteLine("App ApplicationLaunching");
        }


// Code to execute when the application is activated (brought to
foreground)
        // This code will not execute when the application is
first launched
        private void Application_Activated(object sender,
ActivatedEventArgs e)
        {
            Debug.WriteLine("App Application_Activated");
        }


// Code to execute when the application is deactivated (sent to
background)
        // This code will not execute when the application is
closing
        private void Application_Deactivated(object sender,
DeactivatedEventArgs e)
        {
            Debug.WriteLine("App Application_Deactivated");
        }


// Code to execute when the application is closing (eg, user hit
Back)
        // This code will not execute when the application is
deactivated
        private void Application_Closing(object sender,
ClosingEventArgs e)
        {
            Debug.WriteLine("App Application_Closing");
        }
```

LISTING 2 - APP.XAML.CS CODE FRAGMENT MET EVENT HANDLERS VOOR INITIALISATIE EN DE-INITIALISATIE VAN DE APPLICATIE.

Eerst kijken we wat er gebeurt bij opstarten en vervolgens weer afsluiten van de applicatie. Om het verhaal compleet te maken on-

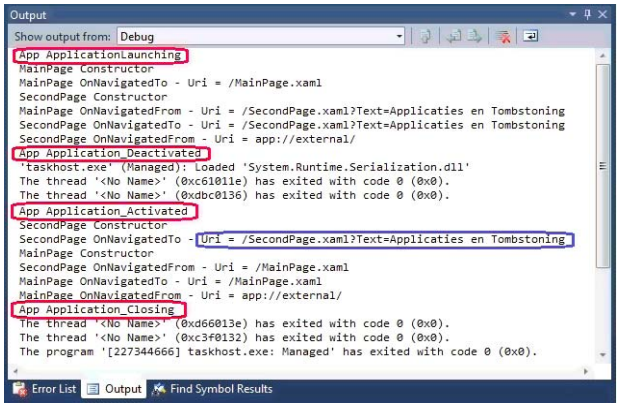
derzoeken we ook direct wat er gebeurt als het tweede scherm zichtbaar wordt. De navigatie is dus als volgt: Bij applicatiestart wordt het hoofdscherm getoond. In de TextBox wordt wat tekst ingevoerd en de fontgrootte wordt met de Slider aangepast. Vervolgens wordt via de Hyperlink genavigeerd naar het tweede scherm. Door tweemaal op de Back toets te drukken keert de applicatie eerst terug naar het hoofdscherm en vervolgens wordt deze afgesloten. Door weer te bekijken welke debug informatie op welk moment wordt getoond is de volgorde van uitvoeren van code goed te bekijken (zie figuur 2).



FIGUUR 2 - LEVENSLOOP VAN DE VOORBEELDAPPLICATIE, AFGESLOTEN DOOR DE BACK TOETS.

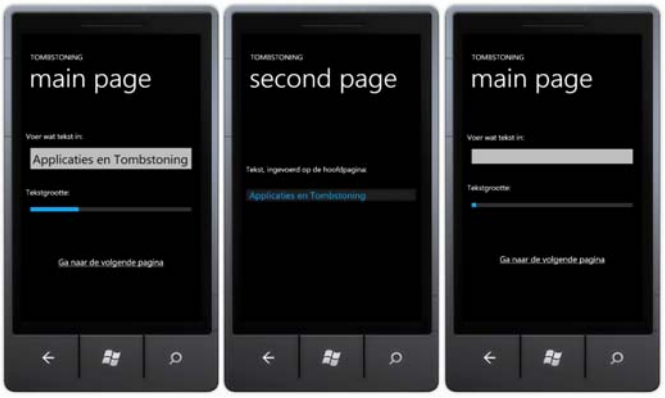
Tussen het ApplicationLaunching event en het Application_Closing event is de applicatie actief. Om informatie aan de gebruiker te tonen hoeft niets bijzonders te worden gedaan. Tijdens navigatie van het hoofdscherm naar het volgende scherm wordt de ingevoerde tekst als parameter meegegeven. Als afwisselend vanuit het hoofdscherm genavigeerd wordt naar het tweede scherm en vervolgens weer terug, wordt telkens voordat het tweede scherm zichtbaar wordt, de constructor voor dat scherm aangeroepen. Met andere woorden: Telkens als naar een nieuw scherm binnen een applicatie wordt genavigeerd, wordt een nieuwe instantie van dat scherm gemaakt.

Vervolgens kijken we wat er gebeurt bij opstarten, navigeren naar het tweede scherm na invoeren van wat tekst, een andere applicatie starten en via de Back toets helemaal teruggaan tot onze applicatie wordt afgesloten (zie figuur 3).



FIGUUR 3 - LEVENSLOOP VAN DE VOORBEELDAPPLICATIE MET TOMBSTONING TUSSENDOOR.

Er zijn nu verschillende momenten waarop de applicatie actief is. Allereerst tussen het ApplicationLaunching event en het Application_Deactivated event. Vervolgens is de applicatie weer actief tussen het Application_Activated event en het Application_Closing event. In de tussentijd was de applicatie niet actief, maar trad Tombstoning op. Bij het opnieuw actief worden van de applicatie valt op dat we direct naar het tweede scherm terugkeren. Dit gedrag is prettig, immers, de gebruiker keert terug naar het scherm in onze applicatie, dat het laatst zichtbaar was. Dit gebeurt zonder dat we daar in de applicatie iets extra's voor hoeven te doen. Terugkeren naar de juiste pagina wordt verzorgd door het besturingssysteem. Ook valt op dat gegevens in de NavigationContext ook nog steeds bestaan (zie het blauwe kader in figuur 5). Vandaar dat de juiste tekst weer wordt getoond in het tweede scherm. Als we echter teruggaan naar het hoofdscherm valt op dat de MainPage constructor wordt aangeroepen en dat we dus een nieuwe instantie krijgen van het scherm. Omdat we niets bijzonders doen, toont dit hoofdscherm dus geen gegevens in de TextBox. Dit kan voor een eindgebruiker verwarrend zijn. Immers, de gegevens die op het tweede scherm nog zichtbaar waren zijn op het hoofdscherm opeens verdwenen. Dit is geïllustreerd in figuur 4.



FIGUUR 4 - GEVOLGEN VAN TOMBSTONING.

Na opstarten van de applicatie en invoeren van wat tekst navigeert de gebruiker naar het tweede scherm. Hier is de eerder ingevoerde tekst zichtbaar. Na verlaten van de applicatie via de Start toets en later weer terugkeren naar de applicatie via de Back toets wordt het tweede scherm in precies dezelfde toestand getoond. Door vervolgens terug te gaan naar het hoofdscherm blijkt dat we informatie zijn kwijtgeraakt. Dit had overigens ook kunnen gebeuren op het tweede scherm, als op dat scherm ook andere UI Controls zoals een TextBox zouden zijn geplaatst.

Oorspronkelijke situatie herstellen

Voor gebruikers van een Windows Phone 7 zou het plezierig zijn als alle applicaties consistent gedrag vertonen. Onderbreken van een applicatie mag bijvoorbeeld niet tot informatieverlies lijden. Dit is de voornaamste reden waarom er extra functionaliteit nodig is binnen alle applicaties. De applicatie zelf moet ervoor zorgen dat (tijdelijk) gegevens moeten worden opgeslagen op het moment dat de applicatie inactief wordt en dus verdwijnt door Tombstoning. De applicatie moet er ook voor zorgen dat eerder opgeslagen gegevens weer zichtbaar worden als een applicatie weer actief wordt vanuit een Tombstoning situatie. Om de benodigde extra functionaliteit goed te implementeren, is het van belang, onderscheid te maken tussen gegevens die permanent moeten worden opgeslagen en gegevens die alleen van belang zijn gedurende de levensduur van de applicatie. Omdat Tombstoning ervoor zorgt dat een applicatie wordt afgesloten, en mogelijk niet via de Back toets opnieuw wordt geactiveerd, is het van belang gegevens die voor het correct werken van de applicatie noodzakelijk zijn, altijd op te slaan als onderdeel van zowel het Application_Deactivated event en het Application_Closing event. Deze

gegevens kunnen bijvoorbeeld efficiënt worden opgeslagen met behulp van serialisatie (al dan niet in combinatie met een BinaryWriter). Het gaat hierbij niet om 'tijdelijke' gegevens die slechts nodig zijn om de applicatie terug te brengen naar de situatie zoals die was toen de applicatie onderbroken werd. Deze laatste soort gegevens kan worden opgeslagen door gebruik te maken van de functionaliteit van de PhoneApplicationService class. Ook is het mogelijk tijdelijke gegevens, specifiek voor een pagina op te slaan in de PhoneApplicationPage. Hier zitten echter wat beperkingen aan voor wat betreft de levensduur van de gegevens en de hoeveelheid gegevens die kunnen worden opgeslagen. In het volgende codefragment wordt uitsluitend gebruik gemaakt van functionaliteit in de PhoneApplicationService class om tijdelijke gegevens op te slaan.

```
protected override void OnNavigatedFrom(System.Windows.Navigation.
NavigationEventArgs e)
{
    base.OnNavigatedFrom(e);
    Debug.WriteLine("MainPage OnNavigatedFrom - Uri = " + e.Uri.
ToString());

    // Overschrijf een eventueel eerder opgeslagen textbox inhoud
    if (PhoneApplicationService.Current.State.
ContainsKey(textBoxContentKey))
    {
        PhoneApplicationService.Current.State.
Remove(textBoxContentKey);
        Debug.WriteLine("MainPage: Removing current TextBox
value");
    }
    PhoneApplicationService.Current.State.Add(textBoxContentKey,
 this.EnteredText.Text);

    // Overschrijf een eventueel eerder opgeslagen slider
posititie
    if (PhoneApplicationService.Current.State.
ContainsKey(sliderPositionKey))
    {
        PhoneApplicationService.Current.State.
Remove(sliderPositionKey);
        Debug.WriteLine("MainPage: Removing current
SliderPosition");
    }

PhoneApplicationService.Current.State.Add(sliderPositionKey,
this.FontSizeSlider.Value);
}

protected override void OnNavigatedTo(System.Windows.Navigation.
NavigationEventArgs e)
{
    base.OnNavigatedTo(e);
    Debug.WriteLine("MainPage OnNavigatedTo - Uri = " +
e.Uri.ToString());

    // Vul de textbox met een eventueel eerder opgeslagen inhoud
    if (PhoneApplicationService.Current.State.
ContainsKey(textBoxContentKey))
    {
        Debug.WriteLine("MainPage: Retrieving stored TextBox
value");

this.EnteredText.Text = (string)PhoneApplicationService.Current.
State[textBoxContentKey];
        NavigationButton.NavigateUri =

new Uri("/SecondPage.xaml?Text=" + this.EnteredText.Text, UriKind.
Relative);
    }

    // Positioneer de slider met een eventueel eerder opgeslagen
positie
    if (PhoneApplicationService.Current.State.
ContainsKey(sliderPositionKey))
    {
        Debug.WriteLine("MainPage: Retrieving stored
SliderPosition");
        this.FontSizeSlider.Value =
(double)PhoneApplicationService.Current.State[sliderPositionKey];
    }
}
```

LISTING 3 - MAINPAGE CODE FRAGMENT VOOR OPSLAAN EN TERUGHALEN VAN GEGEVENS IN TOMBSTONING SITUATIES.

Telkens als een nieuwe instantie van de applicatie wordt gestart, zal de State bibliotheek van de PhoneApplicationService leeg zijn. Iedere keer als we het hoofdscherm verlaten worden de huidige tekst en de huidige Slider positie bewaard. Zodra we terugkeren uit een Tombstoning situatie, zal de State bibliotheek van de PhoneApplicationService gegevens bevatten die bij navigeren naar dit scherm aan de relevante UI Elementen kunnen worden toegekend. Bij starten van een nieuwe applicatie gebeurt dit uiteraard niet.

Een nieuwe applicatie instance starten

Stel dat de gebruiker onze voorbeeldapplicatie start en vervolgens kiest voor het tweede scherm. Daar aangekomen besluit de gebruiker naar het startscherm terug te gaan via de Start toets op de telefoon, waarna onze voorbeeldapplicatie opnieuw opstart vanuit het startscherm. Om dit gedrag zichtbaar te maken moet de applicatie worden gestart vanuit een instantie van Visual Studio 2010. Nadat we terugkeren naar het startscherm moet vervolgens de applicatie weer worden gestart vanuit een tweede Visual Studio 2010 instantie.
Zoals blijkt uit de eerste Visual Studio 2010 instantie, is de gebruiker vanuit de applicatie direct naar het startscherm gegaan, waardoor een Application_Deactivated event wordt veroorzaakt. Er komt echter geen Application_Activated event in de tweede instantie van Visual Studio 2010, maar weer een Application-Launching. In dit geval zal een lege hoofdpagina worden getoond, wat correct gedrag is.

Conclusies

Tombstoning is een vorm van gedrag waar elke Windows Phone 7 ontwikkelaar rekening mee moet houden. Correct reageren op applicaties die al dan niet tijdelijk onderbroken worden hoeft niet altijd moeilijk te zijn. Belangrijk is wel te snappen hoe applicaties op Windows Phone 7 zich gedragen. In eerste instantie lijkt het misschien dat Tombstoning alleen maar extra complexiteit oplevert voor ontwikkelaars. Dit is echter niet waar, omdat alleen de applicatie kan beslissen welke gegevens relevant zijn om tijdelijk te worden opgeslagen. Tegelijkertijd is Tombstoning als mechanisme een vorm van gedrag die voor eindgebruikers onzichtbaar hoort te zijn, en ervoor zorgt dat een Windows Phone 7 altijd snel reageert op alles wat een gebruiker met het apparaat wil doen.

Maarten Struys, is Windows Embedded / Windows Phone Evangelist bij Alten PTS.