# Het ontwikkelen van MS Surface-applicaties

## De weg naar vergaande mens-computer interactie

Toine de Boer

De MS Surface is een van de meest vooruitstrevende NUI (Natural User Interface) apparaten. NUI is de volgende generatie in computer user interfaces, waarbij het niet meer hoofdzakelijk gaat om het uitvoeren van een bepaalde taak maar ook om de ervaring tijdens het uitvoeren van een taak, de zogenaamde UX (User Experience). Een juiste NUI applicatie werkt vanzelfsprekend, intuïtief, direct en moet de gebruiker verleiden om te worden gebruikt en te blijven gebruiken.

In dit artikel worden de tools beschreven die nodig zijn voor MS Surface applicatie-ontwikkeling, vanaf het bouwen van een MS Surface-applicatie tot aan het werkend krijgen van een applicatie op een MS Surface unit. Tevens wordt behandeld hoe vingers en objecten door de MS Surface unit worden verwerkt en hoe dit in de applicatie kan worden afgehandeld.
De afhandeling van vingers en objecten in een applicatie wordt in dit artikel op twee verschillende manieren beschreven, gebruikmakend van een standaard MS Surface control (ScatterView) en het zelf maken van controls voor de MS Surface (Custom Controls). Deze twee beschrijvingen zijn verder uitgewerkt in de twee projecten in de bijbehorende codevoorbeelden bij dit artikel, dat basiscode bevat maar ook meer geavanceerde code.

Het ontwikkelen van applicaties voor de MS Surface gaat grotendeels op dezelfde manier als bij reguliere moderne Windows-applicaties. In beide gevallen kan er gebruik gemaakt worden van XNA en WPF en voor het ontwikkelen van een hoogwaardig grafische en interactieve interface (dit artikel is geschreven voor WPF, maar bevat ook onderdelen relevant voor XNA).
Het horizontaal gepositioneerde scherm van de MS Surface vormt de onderscheidende factor in het ontwerp van MS Surface-applicaties ten opzichte van desktopapplicaties. Vanwege de horizontale tafelpositionering van de MS Surface is het een uitstekend apparaat voor sociaal gebruik. Het 'sociaal gebruik' oftewel 'rondom de tafel gebruik' is een van de grootste onderscheidende kenmerken van de MS Surface, ontwikkelaars moeten ervoor zorgen dat deze ervaring behouden blijft gedurende het gebruik van MS Surface-applicaties. Om het maximale uit de MS Surface te halen moet er rekening worden gehouden met de onderscheidende kenmerken van de MS Surface; 'massive multi touch input', 'multi user', 360° UI, handgebaren/gestures en interactie met fysieke objecten.

### De Surface Units

De MS Surface is er in twee verschillende versies; de 'Commercial Unit' en de 'Developer Unit'. Beiden werken in grote lijnen hetzelfde, alleen bevat de 'Developer Unit' developer licenties en extra geïnstalleerde software om op de unit te kunnen ontwikkelen, wat het daarom ook iets duurder maakt.

De MS Surface-units werken op een Windows Vista 32-bit OS. Hierop worden de 'Surface Input' en 'Surface Shell' geladen die de Vista machine omtovert tot een MS Surface-machine. De 'Surface Input' zorgt voor de afhandeling van alle Touches en Tags geplaatst op het scherm . De Surface Shell is een laag bovenop Windows Vista waarmee de normale Windows Vista interface verdwijnt en een nieuwe MS Surface interface wordt gestart. De Surface Shell is de omgeving waar MS Surface volledig onder werkt en waar de zogenaamde 'Attract Application', de 'Launcher' en de MS Surface applicaties onder werken. De Attract Application is de applicatie die direct wordt geladen bij het opstarten van de Shell. De Launcher is het menu waarvan alle beschikbare MS Surface applicaties kunnen worden gestart.

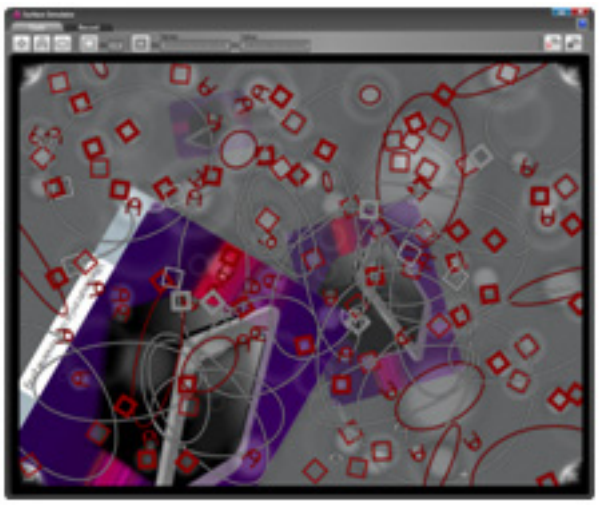
De MS Surface heeft standaard twee useraccounts, het 'admin' en 'TableUser' account. Het adminaccount is bedoeld voor beheerdoeleinden en werkt met een fysiek toetsenbord en muis in een normale Windows Vista-omgeving. Tijdens normaal gebruik werkt de MS Surface hoofdzakelijk onder het 'TableUser' account.
Wanneer de MS Surface werkt onder het 'TableUser' account, worden alle Windows-meldingen niet meer op de unit getoond. Foutmeldingen worden omgezet naar nette MS Surface meldingen en getoond aan een van de zijdes van de MS Surface-unit. De unit wordt direct opgestart naar de Surface Shell zonder enig Windows Vista-laadscherm. Zelfs wanneer de Surface Shell totaal niet meer werkt, worden er geen Windows-schermen getoond

maar alleen een MS Surface 'out of order' melding. Onder het 'TableUser' account krijgt iemand de echte MS Surface-ervaring, waarbij iemand helemaal niet door heeft dat de MS Surface eigenlijk op Windows Vista werkt.

## De MS Surface SDK

Om MS Surface-applicaties te kunnen ontwikkelen is het volgende nodig; de vrij verkrijgbare 'Microsoft MS Surface SDK 1.0 SP1', Visual Studio 2008, .Net 3.5 SP1 en kennis van WPF en/of XNA.
De MS Surface SDK bestaat hoofdzakelijk uit een Surface Simulator (met input voor meerdere USB muizen), extra WPF-controls, VS projecttemplates, Codesamples en Surface-tools waarmee direct met MS Surface applicatie ontwikkeling op een desktop met Vista (32-bit) kan worden begonnen.
De MS Surface SDK geeft verschillende tools die helpen bij het ontwikkelen van MS Surface-applicaties, zoals de Simulator, SurfaceStress en de Tag Generator. Naast de tools geeft de MS Surface SDK ook nog een archief aan uitgebreide codesamples specifiek voor de MS Surface en Help-documentatie waarbij de 'How Do I...? Examples' zeer aan te raden is voor startende MS Surface ontwikkelaars.

De MS Surface SDK geeft de mogelijkheid een MS Surface-project te starten in Visual Studio met projecttemplates voor WPF en XNA. Beide templates bevatten alle bestanden die nodig zijn voor een MS Surface-applicatie om goed te werken op een MS Surface-unit.
Een van de belangrijkste tools in de MS Surface SDK is de Simulator. De Simulator zorgt ervoor dat er op een desktopcomputer kan worden ontwikkeld en getest in een MS Surface-achtige omgeving. Hierin zijn een simulatie van de MS Surface en enkele tools beschikbaar waarmee meerdere vingers, Tags en andere objecten kunnen worden gesimuleerd.
Ook kan direct een extra muis op de computer worden aangesloten die daarna werkt als extra vinger op de Simulator. Voor herhaaldelijke testscenario's beschikt de Simulator over een 'Record' toolbar waarmee de gesimuleerde input kan worden opgenomen en worden afgespeeld.
Omdat er een horizontale user interface wordt ontwikkeld op een verticale user interface is het niet vreemd als de applicatie er niet uit gaat zien als een 360° applicatie, waarbij de applicatie goed en volledig vanaf elke zijde te gebruiken is. Daarom is het verstandig om de user interface van de Simulator op zijn kop te gebruiken door bij het opstarten van de Simulator een van de bovenste twee access points in te drukken, de access points zijn de knoppen in de hoek van de MS Surface. Hiermee start de Launcher en MS Surface applicaties op zijn kop en wordt de ontwikkelaar meer gedwongen om 360° te ontwikkelen.
Daarnaast bevat de SDK een SurfaceStress applicatie waarmee op een MS Surface applicatie, in combinatie met de Simulator, een stresstest kan worden uitgevoerd. Met deze stresstest kan er getest worden op een overvloed aan inkomende vingers, Tags en blobs om te testen hoe de applicatie reageert en of het goed blijft werken bij heel erg veel input.



**FIGUUR1: STRESSTEST OP DE SIMULATOR.**

## Shell & Application Launcher

Omdat ons als kind werd afgeleerd alles aan te raken is er een drempel ontstaan voor het aanraken van openbare touchapparaten. Om deze drempel te verlagen heeft de MS Surface een zogenaamde 'attract application' die mensen ertoe moet zetten - en zelfs moet verleiden - om de MS Surface aan te raken. De User Experience bij een MS Surface begint altijd bij de attract application, standaard is dat het interactieve water. Maar iedereen is vrij om zijn eigen attract application te maken en te implementeren.

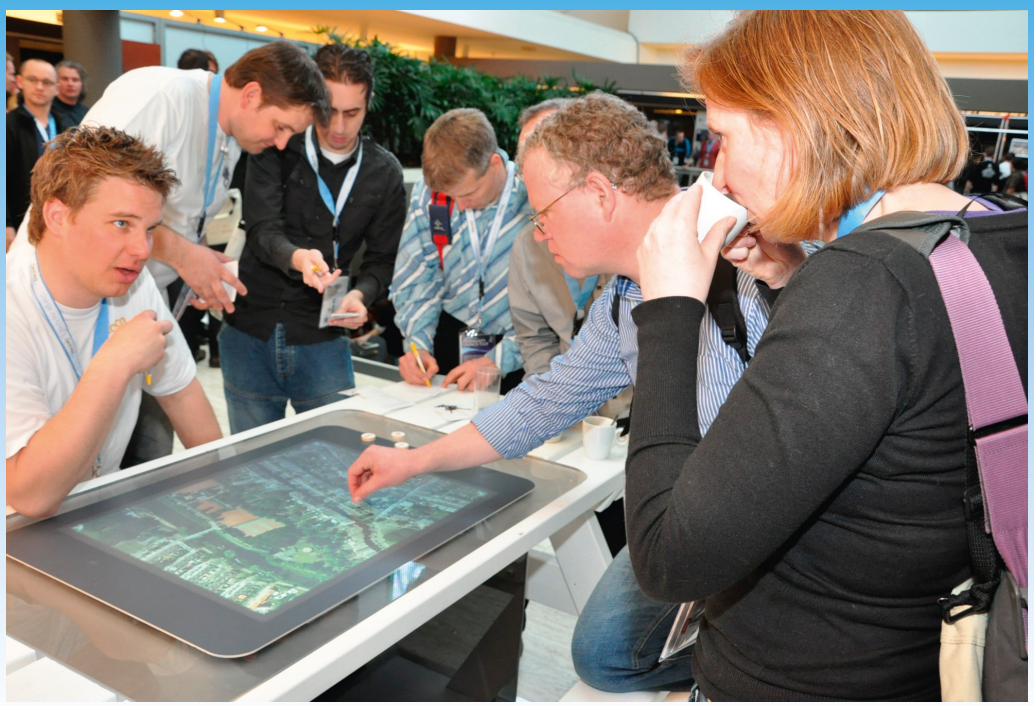
> 'Omdat ons als kind werd afgeleerd alles aan te raken is een drempel ontstaan voor touchapparaten'.

Nadat de MS Surface voor het eerst is aangeraakt verschijnen er knoppen in elke hoek van de MS Surface, de zogenaamde 'access points'. Met deze access points kan er altijd worden geschakeld tussen de 'Launcher' en de applicaties, zoals de startknop in Windows voor de programma's. De Launcher is een horizontaal scrollmenu waar vanuit alle beschikbare applicaties kunnen worden gestart. Voordat een applicatie wordt geopend, wordt er een preview weergegeven van de applicatie, deze kan bestaan uit afbeeldingen, slideshows of een video.

Wanneer een applicatie is opgestart blijven er altijd twee access points zichtbaar waarmee de Launcher weer kan worden geopend. Als vanuit een applicatie de Launcher wordt geopend, blijft de applicatie op de achtergrond geopend. Om andere applicaties volledig van alle MS Surface-resources te kunnen laten benutten is het een goed idee om applicaties te pauzeren wanneer de Launcher wordt geopend en daarna de applicatie weer te hervatten wanneer de focus weer op de applicatie wordt geplaatst.
De Template voor MS Surface-applicaties komt met drie Event handlers die hier goed voor kunnen worden gebruikt. Pauzeer of stop de applicatie bij OnApplicationPreviewed en OnApplicationDeactivated, hervat de werkzaamheden bij OnApplicationActivated.
Om een applicatie in de Launcher getoond te krijgen dient er een XML-bestand geplaatst te worden in het filesysteem in de map '%PROGRAMDATA%\Microsoft\Surface\Programs'. Deze XML wordt meegeleverd in de MS Surface-projecttemplates van Visual Studio en is te vinden in de root van het project. In deze XML-file kunnen een aantal gegevens van de applicatie worden opgegeven, zoals de titel, beschrijving, locatie van de executable, locatie van het icoon en locatie van de previewafbeeldingen of -video.

## Tafelmodel meest gebruikt

Microsoft heeft meerdere multi-touch producten onder de noemer Surface. Hiervan is de 'MS Surface' het vlaggenschip en 'Surface Toolkit for Windows Touch beta' de nieuwste variant. Dit kan verwarrend over komen, wanneer er in dit artikel over de 'MS Surface' gesproken wordt gaat het om de tafelvorm.



```
<?xml version="1.0" encoding="utf-8" ?>
<ss:ApplicationInfo
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:ss="http://schemas.microsoft.com/Surface/2007/
ApplicationMetadata">
  <Application>
    <Title>MySurfaceApp</Title>
    <Description>a ScatterView can handle all objects</
Description>
    <ExecutableFile>%PROGRAMFILES%\YourCompany\YourApp\
ScatterViewExample.exe</ExecutableFile>
    <Arguments></Arguments>
    <!– Afbeelding die in Launcher wordt getoond 250x250 -->
    <IconImageFile>%PROGRAMFILES%\YourCompany\YourApp\Resources\
icon.png</IconImageFile>
    <Preview><!– Preview die in Launcher wordt getoond bij
focus 320x240 -->
  <PreviewImageFile>%PROGRAMFILES%\YourCompany\YourApp\
Resources\iconPreview.png</PreviewImageFile>
    </Preview>
    <!– Geregistreerde tags -->
    <Tags>
      <ByteTag Value="C0">
        <Actions>
          <Launch />
        </Actions>
      </ByteTag>
    </Tags>
  </Application>
</ss:ApplicationInfo>
```

**FIGUUR2: XML VOOR DE LAUNCHER.**

In deze XML kan ook worden aangegeven dat de applicatie moet werken als een attract application. Ook kunnen er Tags worden geregistreerd voor een applicaties, zoals weergegeven in figuur 2 waarbij Tag 'C0' wordt geregistreerd. Een geregistreerde Tag is daarna alleen nog te gebruiken voor de applicatie waarop hij is geregistreerd. Bij het gebruik van de Tag in een andere applicatie zal de Tag een snelkoppeling tonen naar de applicatie waarop hij is geregistreerd.

## MS Surface Input

Een input bij MS Surface wordt een 'contact' genoemd. Een contact wordt door MS Surface onderscheiden in vingers, Tags en alle andere objecten als blobs. Dit wordt volledig door de MS Surface-unit zelf geregeld en kan makkelijk worden achterhaald onder de contactproperties IsFingerRecognized en IsTagRecognized. Een contact kan naast de x- en y-coördinaten van het scherm, met de functie contact.GetCenterPosition, ook de oriëntatie van contact achterhalen met de functie contact.GetOrientation. Hiermee kan worden bepaald waar het contact vandaan komt of hoe deze op de MS Surface unit is geplaatst.

Een contact komt voornamelijk binnen op de ContactDown, ContactChanged en ContactLeave events. Een MS Surface is multi-touch en er moet rekening worden gehouden met het binnenkomen van meerdere contacts tegelijkertijd. Het is dus goed mogelijk dat meerdere contacts tegelijkertijd druk uitoefenen op eenzelfde element. Houd daarom goed bij welke contacts het recht hebben een element te manipuleren en/of er wel meerdere contacts tegelijkertijd dit recht moeten kunnen verkrijgen, voorbeelden hiervan onder 'Custom Controls'. Gelukkig neemt het MS Surface-framework hier het overgrote deel van het werk uit handen. Alle standaard MS Surface-controls zijn werkend gemaakt voor een multi-touch omgeving en de controls reageren grotendeels op een te verwachte manier.

## ScatterView

MS Surface is een geheel nieuw soort user interface, waarbij vooral de eigenschappen multli-touch, tag input en 360° UI naar boven komen. Bij deze nieuwe user interface horen ook nieuwe Controls die speciaal gemaakt zijn voor deze eigenschappen. De MS Surface SDK beschikt over veel bekende WPF-controls die omgebouwd zijn voor Touches zoals de Button, ListBox, Slider en TextBox.

De eenvoudigste manier om input, zoals vingers en objecten, op te vangen en af te handelen is door gebruik te maken van de nieuwe control ScatterView met zijn ScatterViewItems. De ScatterView is een speciaal voor MS Surface ontwikkelde control waarmee direct de onderdelen multi-touch en 360° UI vol benut kunnen worden. Een ScatterView-control is een container om objecten te

tonen op een scherm. ScatterView zorgt ervoor dat alle toegevoegde objecten automatisch zijn te manipuleren zoals het kunnen vergroten en verkleinen, verplaatsen en roteren.
Een ScatterView bestaat uit alleen maar ScatterViewItems maar het accepteert ook andere type objecten, die hij automatisch zelf inkapselt in een ScatterViewItem. In een ScatterViewItem worden alle UI-elementen normaal weergegeven als standaard in WPF. Een niet-UI-element zal worden weergegeven op het scherm in het resultaat van de functie ToString op het object. Dit niet-UI-element kan worden gestyled met een ItemTemplate om het object toch goed te kunnen visualiseren op de MS Surface.
Hoe een ScatterViewItem op de MS Surface kan worden gemanipuleerd kan worden ingesteld met enkele properties zoals CanMove, CanRotate en CanScale (zie figuur 3).
Om een ScatterViewItem eenvoudig te kunnen instellen is het vaak verstandig objecten eerst in een ScatterViewItem te plaatsen, waardoor direct de properties beschikbaar zijn, voordat ze aan de ScatterView worden toegevoegd

```
<s:ScatterView>
    <!– Zonder handmatigge ScatterViewItem -->
    <Image Source=”\Resources\iconPreview.png” />
    <s:ScatterViewItem CanMove=”True” CanRotate=”True”
CanScale=”False”>
  <!– Handmatig Geimplementeerd in een ScatterViewItem -->
        <Image Source=”\Resources\iconPreview.png” />
    </s:ScatterViewItem>
</s:ScatterView>
```

**FIGUUR 3: TWEE MANIEREN OM EEN AFBEELDING TOE TE VOEGEN AAN EEN SCATTERVIEW.**

Omdat objecten op verschillende manieren kunnen worden toegevoegd aan een ScatterView, kunnen de objecten ook op verschillende manieren uit de ScatterView tevoorschijn komen. Als er bijvoorbeeld een image wordt toegevoegd zonder eerst een ScatterViewItem er omheen te plaatsen, dan zal het object in de ScatterView terugkomen als een image, terwijl de ScatterView op de achtergrond toch daaromheen een ScatterViewItem heeft geplaatst. Daarom kan van een object het ScatterViewItem worden achterhaald door de ItemContainerGenerator van de ScatterView, zoals weergegeven in figuur 4. Daarnaast kan het werkelijk toegevoegde object altijd weer worden opgehaald via de Content property van het ScatterViewItem.

```
void svi _ PreviewContactChanged(object sender, ContactEventArgs e)
{
    // ScatterViewItem proberen op te halen
    ScatterViewItem svi = sender as ScatterViewItem;
    if (!(sender is ScatterViewItem))
        // ScatterViewItem uit ItemContainerGenerator ophalen

svi = scv.ItemContainerGenerator.ContainerFromItem(sender) as
 ScatterViewItem;
    // Do some stuff
    svi.Opacity = (svi.ActualWidth + svi.ActualHeight) / 1000;
}
```

**FIGUUR 4: ITEMS OPHALEN UIT SCATTERVIEW.**

De ScatterView kan heel handig zijn, omdat het automatisch zorgt voor veel manipulaties. Daarentegen kan dat ook erg onhandig zijn als iemand zelf animaties wil maken op een ScatterViewItem. Omdat een ScatterViewItem in de achtergrond alle manipulaties en transformaties zelf wil afhandelen kan het ongewenste effecten opleveren wanneer een ontwikkelaar dit ook wil doen in zijn code.
Bij een ScatterViewItem moet daarom nooit direct de property RenderTransform worden gebruikt bij animaties, gebruik hiervoor de Center, Orientation, Height en Width properties. Dit moet daarentegen niet tegelijk worden gebruikt met de manipulaties van het ScatterViewItem zelf. Stop de zelfgemaakte animaties of zet bij de ScatterViewItem tijdelijk de ‘IsHittestVisible’ property op ‘false’, zodat er geen contacts gecaptured kunnen worden en er dus geen automatische Manipulaties meer kunnen voorkomen.

Tevens moet er bij het gebruik van ‘System.Windows.Media.Animation’ animaties voor worden gezorgd dat de animaties goed worden afgerond, zodat het ScatterViewItem weet vanuit welk punt hij zelf verder kan gaan met zijn eigen manipulaties. Om dit te bereiken moeten de geanimeerde properties op hun bereikte eindwaarde worden gezet in het completed event van de animatie. Mocht dit gebeurd zijn dan schiet de ScatterViewItem bij de eerste manipulatie terug naar het punt voor de animatie.

## Custom Controls

De MS Surface SDK wordt geleverd met een lijst aan WPF MS Surface Controls. Dit zijn vooral algemeen bekende WPF Controls werkend gemaakt op touch. Omdat de controls werkend zijn gemaakt voor touch betekent dit niet dat ze ideaal zijn voor het gebruik op de MS Surface, de werkelijkheid is verre daarvan.
De algemene WPF-controls zijn gebaseerd op algemeen GUI gebruik en niet multi-touch en 360° UI. Als een ontwikkelaar over een WPF-control de totale controle wil hebben en volledig gebruik wil maken van multi-touch en de 360° UI, dan moet hij al snel een eigen control maken. Hierbij moet worden gedacht aan het zelf afhandelen van alle binnenkomende contacts en het volledig afhandelen van alle manipulaties.

Om een bestaand WPF-element te laten reageren op een contact kunnen er verschillende Eventhandlers worden gebruikt, OnContactDown om te starten, OnContactChanged om wijzigingen af te handelen en OnContactLeave om een verwijderde contact af te handelen. De EventHandlers kunnen in XAML en in code-behind eenvoudig worden toegevoegd aan een standaard WPF-element zoals figuur 5 in XAML en figuur 6 in code-behind.

```
<Ellipse s:Contacts.ContactDown=”Ellipse _ ContactDown”
         s:Contacts.ContactChanged=”Ellipse _ ContactChanged”
         s:Contacts.ContactLeave=”Ellipse _ ContactLeave” />
```

**FIGUUR 5: CONTACT EVENTS IN XAML.**

```
Contacts.AddContactDownHandler(ellipse, Ellipse _ ContactDown);
Contacts.AddContactChangedHandler(ellipse, Ellipse _ ContactChan-
ged);
Contacts.AddContactLeaveHandler(ellipse, Ellipse _ ContactLeave);
```

**FIGUUR 6: CONTACT EVENTS IN CODE-BEHIND.**

Om de contact te koppelen aan het element waarop het als eerst binnenkomt, kan een contact door een WPF-Element zogenaamd worden *ge-Captured* (zie figuur 7). Hiermee kan eenvoudig vanaf de contact worden achterhaald worden welk Element het contact heeft ge-Captured. Bij standaard WPF MS Surface Controls kunnen verschillende contact lijsten worden opgehaald met ContactsCaptured en ContactsCapturedWithin.

```
// Contact komt voor het eerst op het scherm
private void Ellipse _ ContactDown(object sender,
ContactEventArgs e)
{
    // Object ophalen waar contact op binnen komt
```

```
    Ellipse ellipse = sender as Ellipse;
    // Object Captured het contact
    e.Contact.Capture(ellipse);
    // Ophalen locatie van contact
    Point lastContactPoint = e.Contact.GetPosition(ellipse.Parent
as Canvas);
    // Opslaan van gegevens in het contact
    e.Contact.SetUserData("lastContactPoint", lastContactPoint);
    // Event is afgehandeld
    e.Handled = true;
}

// Contact beweegt over het scherm
private void Ellipse _ ContactChanged(object sender,
ContactEventArgs e)
{
    // Ophalen van gegevens uit het contact
    // (deze zijn eerder handmatig opgeslagen in het contact)
    Object lastContactObject = e.Contact.GetUserData
("lastContactPoint");
    Point lastContactPoint = (Point)lastContactObject;
    // todo: Manipulaties op het Object die contact heeft
ge-Captured
```

**FIGUUR 7: GEGEVENS BEWAREN IN CONTACT.**

Omdat een enkel element soms meerdere contacts tegelijk te behandelen heeft, is het wel eens noodzakelijk om gegevens per contact bij te houden. Een contact biedt de mogelijkheid om gegevens te bewaren en op te halen met de twee functies SetUserData en GetUserData. Deze gegevens gaan wel verloren zodra de betreffende contact van het scherm verdwijnt.
Normale WPF-controls zijn gewend een input tegelijkertijd af te handelen. Voor het afhandelen van meerdere contacts zijn hulpmechanismes gemaakt om hierin te ondersteunen. In de Affine2DManipulationProcessor kunnen meerdere contacts worden geregistreerd waarmee er automatisch verschillende berekeningen worden uitgevoerd, die erg van pas kunnen komen bij het afhandelen van Touches relatief van elkaar. Hierbij gaat het voornamelijk om bewegingen van een Touch ten opzichte van een andere Touch.
In Figuur 8 is er een rectangle gebruikt die alleen met meerdere contacts tegelijkertijd kan worden gemanipuleerd. Bij OnContactDown komen een voor een de contacts binnen. Hierbij worden ze eerst ge-Captured op de rectangle waarna ze worden toegevoegd aan de manipulationProcessor. De manipulationProcessor is zo ingesteld dat het alleen RotationDelta berekent en deze bij veranderingen doorgeeft aan OnManipulationDelta, waarin een Rotatie animatie wordt uitgevoerd op de rectangle met de berekende RotationDelta.

```
Affine2DManipulationProcessor manipulationProcessor;
RotateTransform recRotateTransform;

protected void InitializeSurfaceManipulations()
{
    // Instellen Transformeer instelligen op een FrameWorkElement
    recRotateTransform = new RotateTransform(0, 0, 0);
    multipleContactRectangle.RenderTransform = recRotateTransform;
    multipleContactRectangle.RenderTransformOrigin =
new Point(0.5, 0.5);

    // Initializeren SurfaceManipulations
    manipulationProcessor = new Affine2DManipulationProcessor
        (Affine2DManipulations.Rotate, multipleContactRectangle);
    manipulationProcessor.Affine2DManipulationDelta +=
OnManipulationDelta;
}

// ManipulationProcessor detecteert bewegingen
private void OnManipulationDelta(object sender,
Affine2DOperationDeltaEventArgs e)
{
    // Draai het FrameWorkElement
    // (ahv de draai factor van de contacts)
    recRotateTransform.Angle += e.RotationDelta;
}

// Contact komt voor het eerst op het scherm
protected override void OnContactDown(ContactEventArgs e)
{
    // Capture het contact
    e.Contact.Capture(multipleContactRectangle);
    // Begin het volgen van de contact,
    // (tov. andere contacts die worden gevolgd)
    manipulationProcessor.BeginTrack(e.Contact);
}
```

**FIGUUR 8: MANIPULEREN MET MEERDERE CONTACTS.**

Voor de beste MS Surface-ervaring is het bij MS Surface-applicaties van belang dat elementen zich zo natuurlijk mogelijk gedragen. Het is niet gewenst dat een manipulatie direct wordt afgebroken, en daardoor alles direct tot stilstand komt, wanneer een contact verdwijnt tijdens een manipulatie. Om bij het verwijderen van een contact de manipulatie van een element zo natuurlijk mogelijk te laten verlopen is de Affine2DInertiaProcessor erg aan te raden om te gebruiken bij zelfgemaakt Controls. Het gebruik van de Affine2DInertiaProcessor kan er bijvoorbeeld voor zorgen een natuurlijke vertraging te geven aan een rollende knikker die langzaam tot stilstand komt.

## Tot slot

Met de MS Surface komen we een stap dichterbij de automatisering waarbij de mens zich niet meer aan de automatisering moet aanpassen. Er komt een steeds natuurlijkere mens-computer interactie, en een MS Surface-ontwikkelaar is daardoor veel tijd bezig met IxD (Interaction Design). NUI is nog jong en veel goede NUI-toepassingen, waaronder MS Surface-applicaties, zijn er nog niet gemaakt. Enkele valkuilen zijn het direct gebruiken van bestaande desktop applicaties als MS Surface applicatie, het gebruikend van MS Surface als internet browser, het gebruiken van veel knoppen en teksten alsof het een desktop applicatie is.

Daarom is het handig enkele punten goed in gedachten te houden bij het ontwikkelen van MS Surface applicaties:

- kan de applicatie door meerdere personen tegelijkertijd worden gebruikt;
- kan de applicatie vanaf elke kant van de tafel worden gebruikt;
- werkt de applicatie direct en intuïtief, dus zonder uitleg;
- werk de applicatie direct op/met de content, zonder veel knoppen en menu's.

Met MS Surface kan een ontwikkelaar gebruikers bovennatuurlijk krachten geven, bijvoorbeeld bij het vergroten en verkleinen van foto's. Hierbij moet de ontwikkelaar er goed op letten dat de applicatie niet te bovennatuurlijk wordt en moeilijk wordt te begrijpen. Als ontwikkelaar is het wel goed om "outside the box" te denken, maar wellicht is het nog beter om "on the edge of the box" te denken en niet te ver door te schieten in het bovennatuurlijke.

**Toine de Boer,** is software Engineer bij Logica. Hij is te bereiken via t.de.boer@logica.com en via blog.toinedeboer.nl