Hello World!
derp

This is a title!

kgggggggggggg jdfjdfjjdjsdfjdfs, sjsdfjsdfjsdfjdf,dsfjdfjdfjsdfsdfj. jdfjdfjdfjdfjsdfjsdf efsdfsdfsddf

lijn1 hdfjdsdfjsdfj ssdjdjjsdj.
lijn2

Hello World!

derp

This is a title!

kgggggggggggg jdfjdfjjdjsdfjdfs, sjsdfjsdfjsdfjdf,dsfjdfjdfjsdfsdfj. jdfjdfjdfjdfjsdfjsdf efsdfsdfsddf

lijn1 hdfjdsdfjsdfj ssdjdjjsdj.
lijn2

Programmer solutions

Jef Daels

Jef Daels

Inhoud

1 Cursus aanpassingen **11**

2 Random topics **13**
2.1 Extention methods . . . 13
2.1.1 Gebruik van Extention methods . . . 14
2.1.1.1 LINQ en extention methods . . . 14
2.1.1.2 Animaties en extention methods . . . 14
2.1.2 Extention method surprises . . . 14
2.2 Partial classes . . . 14
2.3 Partial methods . . . 14
2.4 Transformations . . . 14
2.5 Awaitable methodes . . . 15
2.6 (XML) serialisatie in .NET . . . 15
2.7 Var keyword . . . 16
2.8 TryParse methodes . . . 16
2.9 Dynamic keyword . . . 16
2.9.1 lock en volatile keywords . . . 16
2.9.2 WPF toolkit . . . 16
2.9.3 Application settings in WPF . . . 16
2.9.4 conditional[debug] . . . 16
2.9.5 custom build . . . 16
2.9.6 sys tools . . . 16

3 Op aanvraag: Data Binding in WPF **17**
3.1 De klasse nMCTDocent . . . 17
3.2 DisplayMemberPath: the old way . . . 18
3.3 DataContext: default binding source . . . 19
3.3.1 DataContext: duiding . . . 19
3.3.2 Datacontext: instellen op een window . . . 20
3.3.2.1 ViewModel voor de demo . . . 20
3.3.2.2 Instellen van de Window DataContext . . . 20
3.3.2.3 Binden van de ItemsControl.ItemsSource Property . . . 21
3.3.2.4 Instellen van de ItemsControl.ItemTemplate Property . . . 22
3.4 Niet DataContext binding sources . . . 23
3.5 TypeConverter . . . 24

4 Delegates **25**
4.1 Delegates: duiding . . . 25
4.1.1 Relevante namespace . . . 25
4.2 Delegates: technische intro . . . 26
4.2.1 Demo-code en toelichting . . . 26
4.2.2 Covariance en contravariance . . . 27
4.2.2.1 Covariance . . . 27
4.2.2.2 Contravariance . . . 27
4.3 Delegates en events . . . 27
4.3.1 Definitie van een event . . . 28
4.3.2 Abonneren (to subscribe) op een event . . . 28
4.3.2.1 Event subscription: verschil met VB.NET . . . 29
4.3.3 Afvuren van een event . . . 29
4.3.3.1 Afvuren van een event: verschil met VB.NET . . . 29
4.3.4 Meerdere clients voor hetzelfde event . . . 30
4.4 Event Guidelines! . . . 30
4.5 Omtrent Events: . . . 31
4.5.1 Static events . . . 31
4.5.2 Delegates of Interfaces: .NET of JAVA events . . . 32
4.5.3 Routed events . . . 32
4.6 Delegates en multithreading . . . 33
4.6.1 Multithreading: wat bedoelt u? . . . 33
4.6.2 Multithreading: iets voor ons? . . . 33
4.6.3 Asynchrone processing en delegates . . . 34
4.6.3.1 Asynchroon, geen AsyncCallback . . . 34
4.6.3.2 Asynchroon, met AsyncCallback, zonder UI . . . 35
4.6.3.3 Asynchroon, met AsyncCallback, met UI . . . 36
4.6.4 Multithreading met de Thread klasse . . . 36
4.6.5 Event based Ansynchronous Pattern (EAP) . . . 37
4.6.6 Asynchroon programmeren made easy . . . 37
4.6.7 CancellationToken . . . 38
4.6.8 Thread.Sleep . . . 38
4.6.9 Meer info? . . . 38
4.7 Code plumbing, aka 'the strategy pattern' . . . 39
4.7.1 Codeplumbing: een XMLViewer . . . 39
4.7.2 Codeplumbing pitfalls . . . 40
4.8 Delegates: faits divers . . . 41
4.8.1 Action en Func . . . 41
4.8.1.1 Action . . . 41
4.8.1.2 Func . . . 41
4.8.2 Meer dan (begin)-invoke . . . 41
4.8.3 Timers . . . 41
4.8.4 Asynchrone eventverwerking . . . 42
4.8.5 Anonymous methods (.NET 2.0) . . . 42

Jef Daels

4.8.6 Lambda expressions (.NET 3.0) . . . 42
4.9 Delegates: niet voor elk programma een geschikte oplossing . . . 43
4.10 Delegates: opgaves . . . 43

5 Reflection 45
5.1 Reflection: duiding . . . 45
5.1.1 Relevante namespaces . . . 45
5.2 Reflection: attributen voorbeeld . . . 46
5.3 Reflection in .NET: meer dan attributen . . . 47
5.3.1 Reflection in .NET: start . . . 47
5.3.2 Hoe een instantie maken van een klasse in een assembly? . . . 48
5.3.3 .NET en attributes . . . 49
5.3.3.1 .NET en attribute informatie opvragen . . . 49
5.3.3.2 Hoe zelf Attribute klasses maken in .NET? . . . 49
5.4 Een eigen plugin- infrastructuur! . . . 50
5.4.1 Een eigen plugin- infrastructuur: vereisten . . . 50
5.4.2 Een eigen plugin- infrastructuur: mogelijke oplossing . . . 50
5.4.3 Een eigen plugin- infrastructuur: implementatie . . . 51
5.4.4 Eigen plugin infrastructuur: demo . . . 52
5.5 Namespaces met interessante attributen . . . 52
5.6 Reflection: opgaves . . . 52

6 Runtime compilatie 53
6.1 Runtime compilatie: duiding . . . 53
6.1.1 Relevante namespaces . . . 53
6.2 .NET compilaties starten . . . 54
6.2.1 Command line compilatie . . . 54
6.2.1.1 Command line compilatie: $C^{\#}$ . . . 54
6.2.1.2 Command line compilatie: VB.NET . . . 55
6.2.1.3 Een extraatje omtrent MSBuild . . . 56
6.2.2 Runtime compilatie: System.CodeDom.Compiler . . . 57
6.3 CodeDom . . . 58
6.4 System.Reflection.emit . . . 58
6.5 Het Roslyn project . . . 58
6.6 Assembly code inspecteren . . . 59
6.6.1 ILDasm . . . 59
6.6.2 ILSpy . . . 60
6.6.3 Reflector . . . 61
6.6.4 Obfuscatie . . . 62
6.6.4.1 Confuser . . . 62
6.7 Opgaves . . . 63
6.7.1 Een tekenprogramma . . . 63
6.7.2 ILSpy . . . 63
6.7.3 Obfuscatie . . . 63

7 Backtracking **65**
7.1 Backtracking: duiding . . . 65
7.1.1 Relevante namespaces . . . 65
7.1.2 Backtracking: principe . . . 65
7.2 N-Queens probleem . . . 66
7.2.1 Probleemstelling, synchrone oplossing . . . 66
7.2.2 Asynchroon: BeginInvoke/4 . . . 66
7.2.3 Asynchroon: async/await . . . 67
7.3 Sudoku's oplossen . . . 67
7.3.1 Sudoku: spelregels . . . 67
7.3.2 Software vereisten . . . 68
7.3.3 Sudoku oplos algoritme . . . 68
7.3.3.1 Constraint satisfaction solver . . . 69
7.4 Sudoku routed events introductie . . . 69
7.5 Async methodes: TaskCompletionSource . . . 70
7.5.1 Task- klasses . . . 71
7.5.2 TaskCompletionSource- klasses . . . 71
7.6 Het MiniMax algoritme (geen examen stof) . . . 73
7.6.1 MiniMax benadering voor vier op een rij . . . 73
7.6.1.1 BordWaarde . . . 73
7.6.1.2 BesteZet . . . 74
7.6.1.3 MiniMax methode . . . 75
7.6.2 Minimax optimalisaties . . . 75
7.6.2.1 Alfa-Beta pruning . . . 75
7.6.2.2 Multi- threading . . . 76
7.7 Backtracking opgaves . . . 77

8 Dependency properties **79**
8.1 Dependency properties: duiding . . . 79
8.1.1 Relevante namespaces . . . 79
8.2 F1URL: een help infrastructuur . . . 80
8.2.1 F1URL gebruik in XAML . . . 80
8.2.2 F1URL definitie . . . 81
8.2.3 F1URL infrastructuur . . . 82
8.3 Dependency properties: technisch . . . 83
8.4 Dependency property XAML binding . . . 84
8.5 Dependency property syntax . . . 84
8.6 Dependency properties en animaties . . . 85
8.6.1 DoubleAnimation . . . 85
8.6.2 ColorAnimation . . . 86
8.6.3 Transformatie animaties . . . 87
8.6.4 Animaties en XAML . . . 88
8.6.5 $C^{\#}$of XAML animaties? . . . 88
8.7 DependencyProperties in XAML: TypeConverters . . . 89

Jeff Baeis

8.7.1 IValueConverter interface . . . 89
8.7.2 Default dataconverters . . . 90
8.7.2.1 TryParse . . . 90
8.8 Dependency properties en OnRender . . . 91
8.9 Veel voorkomende fouten . . . 91
8.10 Extra lectuur . . . 91
8.11 Dependency property: opgaves . . . 92

9 Templated controls 93
9.1 Templated controls: duiding . . . 93
9.2 Relevante namespaces . . . 93
9.3 CaptionControl . . . 94
9.3.1 PARTS . . . 94
9.3.1.1 Caption control via PARTS: inleiding . . . 94
9.3.1.2 Caption Control via PARTS: uitwerking . . . 95
9.3.1.3 Caption control via Parts: test en aanpassing . . . 96
9.3.1.4 Caption Control via PARTS: evaluatie . . . 97
9.4 Caption Control opgave . . . 97
9.5 States . . . 97
9.6 State opgave . . . 99

10 Behaviors 101
10.1 Behaviors: duiding . . . 101
10.2 Relevante namespaces . . . 101
10.3 Drag behavior . . . 102
10.3.1 Drag behavior: dependency property versie . . . 102
10.3.2 Drag behavior: Blend out of the box versie . . . 102
10.3.3 Drag behavior: gespecialiseerde Blend versie . . . 103
10.4 Grip behavior . . . 103
10.5 Resize behavior . . . 104
10.5.1 Alignment impact (Laurenz Demey 2010- 2011) . . . 104
10.6 Behavior opgaves . . . 105
10.7 WPF en application settings . . . 105
10.8 Enumerations: getalwaardes en bewerkingen . . . 107
10.9 Extra lectuur . . . 108

11 iTextSharp 109
11.1 Installatie . . . 109
11.2 PDF creatie (en opgaves) . . . 109
11.3 Manipulatie van een bestaande PDF (en opgaves) . . . 110
11.4 Tekenen (en opgaves) . . . 110
11.5 Toepassingsgebied voor iText . . . 111

12 SQLServer en .NET integratie **113**
12.1 Nodige softwares . . . 113
12.2 Kwadraten berekenen . . . 114
12.3 SQL Server tabellen raadplegen . . . 115
12.4 Debugging SQLServer . . . 116
12.4.1 Debugging startend vanuit het Visual Studio project . . . 116
12.4.2 Debugging van een SQL Server call gestart buiten het project . . . 116
12.5 Theorie vragen . . . 117

13 Windows installer oplossingen **119**
13.1 Demo programma . . . 119
13.2 InstallShield Limited Edition . . . 120
13.2.1 Download en installatie . . . 120
13.2.2 Aanmaken van een MSI-pakket . . . 120
13.2.3 Redistributables . . . 120
13.2.4 Evaluatie InstallShield Limited Edition . . . 120
13.3 WiX . . . 121
13.3.1 Download en installatie . . . 121
13.3.2 Aanmaken van een MSI-pakket . . . 121
13.3.3 Redistributables . . . 121
13.3.4 Evaluatie WiX . . . 121
13.4 Welke installer? . . . 122
13.4.1 Vergelijking InstallShield en Wix . . . 122
13.4.2 Vergeet niet: . . . 122
13.5 Voorbeeld labo examen opgave . . . 122

14 AOP: Aspect Oriented programming **123**
14.1 AOP duiding . . . 123
14.1.1 Relevante namespaces . . . 123
14.2 PostSharp . . . 124
14.2.1 PostSharp kennismaking: methode-entry teller . . . 124
14.2.2 PostSharp en parameter annotaties . . . 125
14.2.3 PostSharp en performance counters . . . 127
14.2.3.1 Eigen performance counters . . . 128
14.2.3.2 AOP en performance counters . . . 128
14.3 AOP oefeningen . . . 129

15 Query Notification **131**
15.1 Query notification oefening . . . 131

16 Discussie topics **133**
16.1 Duiding . . . 133
16.2 Sinus berekenen mbv Taylor reeks . . . 134

Jef Daels

17 Global Assembly Cache: GAC **135**
17.1 Global Assembly Cache: duiding . . . 135
17.1.1 Relevante namespaces . . . 135
17.2 GAC: technische duiding . . . 136
17.3 Ontwikkelen voor en met de GAC . . . 138
17.3.1 Strong named assemblies . . . 138
17.3.2 Gacutil.exe . . . 139
17.3.3 DLL-versies maken . . . 140
17.3.4 Redirecting assemblies . . . 141
17.3.4.1 Welke assembly wordt geladen? . . . 141
17.3.4.2 Redirecting met configuration file . . . 142
17.3.4.3 Redirecting met policy files . . . 143
17.3.4.4 Redirecting op machine niveau . . . 144
17.3.4.5 Redirecting redirections . . . 144
17.3.5 GAC en project references . . . 144
17.3.6 Installeren in de GAC vanuit de Setup (tem VS 2010) . . . 144
17.4 GAC opgave . . . 145
17.5 Extras . . . 146

18 Visual Studio geautomatiseerde broncode manipulaties **147**
18.1 Visual Studio geautomatiseerde broncode manipulaties: duiding . . . 147
18.2 Relevante namespaces . . . 147
18.3 Toolbox codeblocks . . . 148
18.3.1 Voordeel van toolboxcode . . . 148
18.3.2 Nadelen van toolboxcode . . . 148
18.4 Code snippets . . . 149
18.4.1 Snippets gebruiken . . . 149
18.4.2 Eigen code snippets . . . 150
18.4.3 Voordelen van snippets . . . 150
18.4.4 Nadeel van snippets . . . 150
18.5 Macro's in Visual Studio . . . 151
18.5.1 Macro samples . . . 151
18.5.2 Eigen macro's . . . 151
18.5.3 Voordelen van macro's . . . 151
18.5.4 Nadeel van macro's . . . 151
18.6 Visual Studio Addin's . . . 152
18.6.1 Creatie van een Visual Studio Add-in . . . 152
18.6.2 Testuivoering van een Add-in . . . 152
18.6.3 Een lijnenteller Add-In . . . 153
18.6.4 Reference historiek Add-In . . . 153
18.7 Visual Studio events . . . 155
18.8 Deployment van een Add-In . . . 156
18.9 Visual Studio en MEF: extentions . . . 157
18.9.1 cmnt extention . . . 157

18.9.2 cmt extention implementatie 157
18.9.3 cmt deployment 158
18.9.4 Extentions en Visual Studio 158
18.10 Relevante literatuur 158
18.11 Codemanipulatie opgaves 159

Jef Daels

Hoofdstuk 1

Cursus aanpassingen

- templated controls: een no-code kalender (2 views: 3 cbo's, en een maandgrid met navigatie)
- content uit sexy $C^{\#}$

Jef Daels

Hoofdstuk 2

Random topics

2.1 Extention methods

Extention methods bieden de mogelijkheid om, op syntactische wijze, methodes toe te voegen aan klassen: u kan met behulp van de .- notatie deze methodes oproepen. Dit is enkel *syntactic sugar*, omdat de klasse definitie zelf niet wordt uitgebreid (hiervoor zou u overerving nodig hebben).

Het definiëren van een extention method gebeurd door middel van een static routine, zoals hieronder met een voorbeeld beschreven:

```
public static int AantalKlinkers(this string tekst)
    {
    int iAantal = 0;
    foreach (char c in tekst)
        if (IsKlinker(c)) iAantal++;
    return iAantal;
    }
```

Figuur 2.1: Extention method definition

Bij het coderen van een extention method zal de intellisense tonen dat het een extention method betreft:

```
public MainWindow()
    {
    InitializeComponent();
    Console.WriteLine("aantal klinkers in " + this.Title + ": "
        + this.Title.AantalKlinkers());
    int i = this.Title.aantal
    }
}
```

AantalKlinkers | (extension) int string.AantalKlinkers()
All | Tokens

Figuur 2.2: Extention method gebruik

2.1.1 Gebruik van Extention methods

2.1.1.1 LINQ en extention methods

Extention methods werden geïntroduceerd tesamen met LINQ, wiens implementatie grotendeels gebaseerd is op Extention methods (zie de intellisense bij het oproepen van een LINQ-methode)

2.1.1.2 Animaties en extention methods

Designers zullen animaties beschrijven door middel van XAML code (bijvoorbeeld bij een STATE-wijziging bij het gebruik van de VisualStateManager). Eenzelfde animatie zal op verschillende plaatsen gedupliceerd worden. Indien we de animatie als extention method implementeren kan ze eenvoudig (via code) herhaaldelijk worden toegepast:

```
spNumbers11.FadeTo((speler1Waits) ? 0.15 : 1.0, 500);
spNumbers12.FadeTo((speler1Waits) ? 0.15 : 1.0, 500);
spNumbers21.FadeTo((!speler1Waits) ? 0.15 : 1.0, 500);
spNumbers22.FadeTo((!speler1Waits) ? 0.15 : 1.0, 500);
```

Figuur 2.3: Extention method gebruik: fadeto

Opmerking: de gecodeerde animatie is gemakkelijk in een behavior te plaatsen zodat ze ook vanuit XAML snel kan worden geactiveerd.

Jef Daels

2.1.2 Extention method surprises

Extention methodes gedefinieerd op een klasse zijn ook van toepassing op instanties van een ervende klasse.

2.2 Partial classes

2.3 Partial methods

2.4 Transformations

Een transformatie kan opgebouwd zijn uit verschillende elementen: een draaiing (rotatie) een schaling (scaling), een verplaatsing (translation) en kanteling (skewing). Indien u een tranformatie op een object toepast dient u rekening te houden met de reeds bestaande transformaties: indien u een object wenst te herschalen mag u de eventuele draaiing niet verwijderen. Een CompositeTransform object zal in veel gevallen uitermate geschikt zijn om met één object al uw tranformaties te defini¨eren en animeren. Dit vermijdt dat u voor een animatie alle TransformGroup onderdelen moet nakijken om de gepaste tranformatie te vinden.

2.5 Awaitable methodes

.NET 4.5 introduceert *awaitable* methodes. Deze methodes zijn herkenbaar aan de suffix *Async*. Hun uitvoering zal (normaal gezien :)) door het keyword *await* worden voorafgegaan. In dat geval zal de rest van de code pas worden uitgevoerd nadat de (asynchrone) oproep van de Async-methode is afgelopen (dit wordt meer in detail toegelicht in het hoofdstuk omtrent delegates en het hoofdstuk omtrent backtracking).

```
public static async Task XMLSerialise(Bord brd)
    {
    StorageFolder storageFolder = ApplicationData.Current.LocalFolder;
    StorageFile sf = await storageFolder.CreateFileAsync("numberlegions
    Stream s = await sf.OpenStreamForWriteAsync();
    XmlSerializer serializer = new XmlSerializer(typeof(Bord));
    serializer.Serialize(s, brd);
    s.Dispose();
    }
```

Figuur 2.4: Await demo

Belangrijk hierbij op te merken zijn volgende elementen:

- indien u een methode *await*, dan kan dit enkel in een methode die als *async* gedefinieerd werd;
- indien een methode als *async* gedefinieerd wordt is ze op haar beurt awaitable.
  - een async methode zal, indien opgeroepen zonder await, asynchroon worden uitgevoerd: de oproepende methode wacht niet op voltooiing vooraleer verder te gaan met zijn logica:

```
        async void brdNumbers_SpelerToggled(object se
            {
            _isToetsVerwerkingBusy = false;
            pSpeler1ScoreA.Text = "" + brdNumbers.Sco
            pSpeler2ScoreA.Text = "" + brdNumbers.Sco
            pSpeler1ScoreB.Text = "" + brdNumbers.Sco
            pSpeler2ScoreB.Text = "" + brdNumbers.Sco
            await Bord.XMLSerialise(brdNumbers);
            await ZetVolgendeZet(e);
            }
```

Figuur 2.5: Await demo async methode

2.6 (XML) serialisatie in .NET

aanbrengen In een context van application life cycle

2.7 Var keyword

als je type kent: gebruik het! var is vooral interessant om het resultaat type van een linq query niet zelf te moeten tikken.

2.8 TryParse methodes

boolean result, twee parameters, tweede parameter is ref, lijkt me zeer toepasbaar bij dataconverters in XAML

2.9 Dynamic keyword

2.9.1 lock en volatile keywords

2.9.2 WPF toolkit

2.9.3 Application settings in WPF

Best way to bind WPF properties to ApplicationSettings in C#?

2.9.4 conditional[debug]

2.9.5 custom build

2.9.6 sys tools

Jef Daels

Hoofdstuk 3

Op aanvraag: Data Binding in WPF

Databinding biedt WPF de mogelijkheid om declaratieve visualisaties voor data te definiëren via XAML-code. Het betrachtte voordeel is het aanpasbaar maken van de voorstelling zonder de achterliggende programmeer code te moeten aanpassen. Dit introduceert bijna automatisch de mogelijkheid om verschillende templates op hetzelfde gegeven toe te passen. In het MVVM-model wordt deze manier van werken sterk gepromoot.

3.1 De klasse nMCTDocent

Dit hoofdstuk betreft het visualiseren van data door middel van bindings. Vooraleer we de bindings toelichten moeten we dan ook een dataklasse opzetten:

```
public class nMCTDocent
    {
    public String FNaam { get; set; }
    public String VNaam { get; set; }
    public string Foto { get; set; }

    public static ObservableCollection<nMCTDocent> Docenten
        {
        get
            {
            ObservableCollection<nMCTDocent> l = new ObservableCollection<nMCTDocent>();
            l.Add(new nMCTDocent() { FNaam = "Gates", VNaam = "Bill", Foto = @"fotos\bgates.png" });
            l.Add(new nMCTDocent() { FNaam = "Knuth", VNaam = "Donald", Foto = @"fotos\dknuth.jpg" });
            l.Add(new nMCTDocent() { FNaam = "Turing", VNaam = "Alan", Foto = @"fotos\aturing.png" });
            l.Add(new nMCTDocent() { FNaam = "Lovelace", VNaam = "Ada", Foto = @"fotos\alovelace.png" });

            return l;
            }
        }
```

Figuur 3.1: De klasse nMCTDocent

- het *ObservableCollection* type wordt verder in deze tekst toegelicht. Momenteel volstaat het om dit type te beschouwen als een soort *List*;

- het is belangrijk dat de docentlijst via een property wordt aangeleverd. Op deze manier zullen we wat verder in de tekst lijsten kunnen binden met de *Docenten*, waarbij elke docent op zijn beurt zal gebind worden aan zijn visualisatie.

3.2 DisplayMemberPath: the old way

Databinding bestond reeds voor het .NET tijdperk. Zo kon men de (tekst-) visualisatie van een data- element in een lijst koppelen aan een property- waarde van dit data-element. De koppeling gebeurde door het *DisplayMember* in te stellen op de lijst. In WPF bestaat deze mogelijkheid ook nog en moeten we het *DisplayMemberPath* voor de lijst instellen:

```
<ComboBox x:Name="cboDocenten" DisplayMemberPath="FNaam"
          Grid.Row="1" Grid.Column="2"  Margin="3" />
```

Figuur 3.2: DisplayMemberPath binding

Een foutieve property-naam bij het binden resulteert niet in een compilatie of runtime- fout: het gegeven wordt gewoon niet gevisualiseerd. U kan dit wel terugvinden in de output van uw programma (u merkt dat de binding case sensitive is):

```
System.Windows.Data Error: 40 : BindingExpression path error: 'fNaam' property not found on 'object' ''nMCTDocent'
System.Windows.Data Error: 40 : BindingExpression path error: 'fNaam' property not found on 'object' ''nMCTDocent'
System.Windows.Data Error: 40 : BindingExpression path error: 'fNaam' property not found on 'object' ''nMCTDocent'
System.Windows.Data Error: 40 : BindingExpression path error: 'fNaam' property not found on 'object' ''nMCTDocent'
```

Figuur 3.3: BindingPathError

- in bovenstaande situatie moet de invulling van de combobox elders ($C^{\#}$, de zogeheten *code behind*) gebeuren. Dit is de voornaamste tekortkoming van deze benadering waardoor de visualisatie sterk afhankelijk van het programmeer werk blijft;

- elk element in de combobox wordt ondervraagd om de waarde van zijn *FNaam* property te bekomen (u weet hoe dit kan via reflection);

- de bekomen waarde (een tekst) wordt getoond als voorstelling van het object. Eigenlijk ben ik verrast dat deze binding enkel teksten kan tonen (u zal verder in de tekst merken dat binding zeker niet beperkt is tot tekst). Onderstaande ILSpy informatie lijkt dit voor wat betreft de *DisplayMemberPath*- situatie echter te bevestigen:

```
// System.Windows.Controls.ItemsControl
private void UpdateDisplayMemberTemplateSelector()
{
    string displayMemberPath = this.DisplayMemberPath;
    string itemStringFormat = this.ItemStringFormat;
    if (string.IsNullOrEmpty(displayMemberPath) && string.IsNullOrEmpty(itemStringFormat))
    {
        if (this.ItemTemplateSelector is DisplayMemberTemplateSelector)
        {
            base.ClearValue(ItemsControl.ItemTemplateSelectorProperty);
        }
        return;
    }
    DataTemplateSelector itemTemplateSelector = this.ItemTemplateSelector;
    if (itemTemplateSelector != null && !(itemTemplateSelector is DisplayMemberTemplateSelector) && (base.ReadLo
    {
        throw new InvalidOperationException(SR.Get("DisplayMemberPathAndItemTemplateSelectorDefined"));
    }
    this.ItemTemplateSelector = new DisplayMemberTemplateSelector(this.DisplayMemberPath, this.ItemStringFormat)
}
```

Figuur 3.4: ILSpy DisplayMemberPath info

3.3 DataContext: default binding source

3.3.1 DataContext: duiding

Opdat een visualisatie een dataelement zou kunnen tonen moet het dit dataelement natuurlijk kennen. In een WPF- context spreken we dan van de *DataContext* van het UI-element. Een goed begrip van deze *DataContext* is cruciaal om databinding goed te begrijpen. Volgende punten zijn belangrijk:

- indien een UI-element zijn eigen datacontext niet expliciet instelt wordt de datacontext van de UI-parent overgenomen;
  - dit is de hoeksteen van het *MVVM*- gebeuren: we maken één static resource die we instellen als datacontext van het window, alle UI-elementen in dat window erven deze datacontext. Het is dan natuurlijk nodig dat elk viseel element een property-tegenhanger heeft in de viewmodel klasse;
  - uitzonderingen zijn items in lijsten, waarbij elke visualisatie van een lijst-element dit lijstelement als datacontext krijgt ingesteld (automatisch);
- een datacontext instellen zonder deze te gebruiken in een binding is geen enkel probleem;
- bij het ontwikkelen van mijn eigen templated controls zal ik in de constructor de control (bijna) altijd als zijn eigen datacontext instellen.

3.3.2 Datacontext: instellen op een window

3.3.2.1 ViewModel voor de demo

Ook nu moeten we een klasse (het viewmodel) opzetten om die later te gebruiken in de visualisatie:

```
public class BindingsDemoVM
    {
    public string WindowTitel {
        get { return "Binding demo"; }
        }
    public ObservableCollection<nMCTDocent> Docenten {
        get { return nMCTDocent.Docenten; }
        }
```

Figuur 3.5: ViewModel klasse

In eerste instantie levert het viewmodel twee properties:

- *WindowTitel* die gebind zal worden aan de Title- property van het Window;
- *Docenten* die een lijst van docenten zal aanleveren, die in een lijstcontrol (Combobox, ListView, GridView, ..) getoond kan worden.

Jef Daels

3.3.2.2 Instellen van de Window DataContext

- in XAML moet een element reeds gedefinieerd zijn vooraleer het gebruikt kan worden. Omdat de resources van een Window na de *Window*- tag komen, en de Title- property in die Window- tag wordt ingesteld is het geen optie om de datacontext in de *Window.Resources* te definiëren. De plaats om de DataContext van het opstartwindow te definiëren is in de resources van de App.xaml- file (deze wordt gelezen voor het Window):

```
<Application.Resources>
    <src:BindingsDemoVM x:Key="vm" />
</Application.Resources>
```

Figuur 3.6: Application Resources

- op basis van de key kan dit (static) resource gebruikt worden in een binding:

```
<Window x:Class="BindingsDemo.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/pr
    Height="350" Width="525"
    Title="{Binding Path=WindowTitel}"
    DataContext="{Binding Source={StaticResource vm}}"
    >
```

Figuur 3.7: Window binding

De window- tag bevat twee bindings:

- het binden van de DataContext aan de static resource *vm*;
  - indien gewenst kan de DataContext ook via code in de constructor worden ingesteld. Dit kan zinnig zijn indien u geen DataContext instantie als resource kan aanmaken;
- het binden van de *Title* property van het window aan de *WindowTitel* property van de datacontext (indien geen source wordt ingesteld op de binding, dan is de datacontext de source);
  - indien de DataContext van de controls in het Window niet wordt overschreven kan nu elke String- property van elke control in het window (ook deze die elders in een template worden beschreven en nog geladen moeten worden) gebind worden aan de *WindowTitel* van de overgenomen datacontext (bemerk dat de datacontext niet expliciet wordt ingesteld):

```
<TextBlock Text="{Binding WindowTitel}" Grid.Row="3" Grid.Column="1" />
```

Figuur 3.8: DataContext binding

3.3.2.3 Binden van de ItemsControl.ItemsSource Property

Indien we lijsten wensen te visualiseren zullen we hiervoor wellicht de ItemsControl klasse (of een afgeleide) gebruiken. Deze klasse heeft een *ItemsSource* property van het (Interface-) type IEnumerable: dit kan zowel een List, een Array, .. zijn, zolang de IEnumerable interface maar geïmplementeerd wordt:

ItemsControl.ItemsSource Property

.NET Framework 4.5 | Other Versions | 0 out of 5 rated this helpful

Gets or sets a collection used to generate the content of the ItemsControl.

Namespace: System.Windows.Controls
Assembly: PresentationFramework (in PresentationFramework.dll)
XMLNS for XAML: http://schemas.microsoft.com/winfx/2006/xaml/presentation, http://schemas.

Syntax

C# | C++ | F# | VB

```
[BindableAttribute(true)]
public IEnumerable ItemsSource { get; set; }
```

Figuur 3.9: ItemsSource declaratie

Indien u in een *C#*- window *F12* drukt op het *ObservableCollection*- type wordt de gekende code getoond:

```
public class ObservableCollection<T> : Collection<T>, INotifyCollectionChanged, INotifyPropertyChanged
```

Figuur 3.10: Observable hoofding

Indien we nu terug *F12* duwen op *Collection* dan bekomen we:

```
public class Collection<T> : IList<T>, ICollection<T>, IList, ICollection, IReadOnlyList<T>, IReadOnlyCollection<T>, IEnumerable<T>, IEn
```

Figuur 3.11: Collection hoofding

Hieruit kunnen we afleiden dat de *ObservableCollection*- klasse, via het erven van de *Collection*- klasse ook de *IEnumerable* interface implementeert, en de *Docenten* property van het modelview dus geschikt is als bindings source van de ItemsSource van een ComboBox:

```
<ComboBox x:Name="cboDocentenBound" Grid.Row="2" Grid.Column="2"
          DisplayMemberPath="FNaam" Margin="3"
          ItemsSource="{Binding Docenten}" />
```

Figuur 3.12: ItemsSource binding

Opdat de combobox zou weten welke informatie van het dataelement getoond moet worden wordt ook nu terug de *DisplayMemberPath* property ingesteld.

3.3.2.4 Instellen van de ItemsControl.ItemTemplate Property

Om een rijke user experience mogelijk te maken willen we natuurlijk meer dan enkel tekst waardes kunnen tonen. Indien u een DataTemplate hebt voor uw te tonen dataelement kan u dit in de *ItemsControl* instellen met behulp van de *ItemTemplate* property die dan wordt toegepast op elk van de items in de *ItemsSource*:

```
<ComboBox x:Name="cboDocentenBoundTemplated" Grid.Row="3" Grid.Column="2"
          Margin="3"
          ItemTemplate="{StaticResource tmplDocenten}"
          ItemsSource="{Binding Docenten}" />
```

Figuur 3.13: ItemTemplate instellen

Bovenstaande code veronderstelt het bestaan van de *tmplDocenten* datatemplate. Deze wordt gedefinieerd in een resource sectie van bijvoorbeeld het window of de app.xaml file:

Jef Dewels

```
<DataTemplate x:Key="tmplDocenten" DataType="src:nMCTDocent" >
    <Grid Margin="4">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" SharedSizeGroup="Key" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <TextBlock Text="{Binding FNaam}" FontWeight="Bold"  />
        <Image Source="{Binding Foto}" Grid.Column="1" />
    </Grid>
</DataTemplate>
```

Figuur 3.14: nMCTDocent datatemplate

Het resultaat van deze binding met deze template ziet er als volgt uit:

Figuur 3.15: nMCTDocenten in een ComboBox

3.4 Niet DataContext binding sources

De DataContext, zoals de naam reeds suggereert, laat toe om in een binding de gegevens van het te visualiseren element (de data) te gebruiken. Deze datacontext is niet altijd de bron van het nodige gegeven. Voorbeelden waar de nodige informatie elders gehaald moet worden zijn:

- in de ccCaption control worden images getoond. We wensen deze als vierkant te tonen, wiens breedte gelijk is aan de hoogte van de Grid waarin ze worden geplaatst;
- bij het ontwikkelen van een custom control wensen we de properties die de designer instelt op de control te gebruiken in de template;

Om bovenstaande gevallen te kunnen afhandelen moet de *binding source* ingesteld worden. Dit kan gebeuren aan de hand van één van drie properties:

- Source: indien u een bepaalde property wenst te binden aan een ander (static) resource dan de huidige datacontext, zonder de datacontext aan te passen

- RelativeSource (MSDN): om de binding source aan te geven relatief ten opzichte van de control die de binding uitvoert. De syntax voor deze source instelling steunt op RelativeSourceMode Enumeration. Googlen zal ook hier uw beste vriendje zijn (lees zeker ook de MSDN entry);
- ElementName: indien u de binding source die u wenst te gebruiken bij naam kan identificeren in de Visual Tree.

Wanneer bijvoorbeeld de breedte van een GridColumn gebind moet worden aan de hoogte van die Grid kan onderstaande XAML-code gebruikt worden:

```
<DataTemplate x:Key="tmplDocenten2" DataType="src:nMCTDocent" >
    <Grid Margin="4" Height="60">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" SharedSizeGroup="Key" />
            <ColumnDefinition
                Width="{Binding RelativeSource={RelativeSource FindAncestor,
                AncestorType={x:Type Grid}}, Path=ActualHeight}"/>
        </Grid.ColumnDefinitions>
        <TextBlock Text="{Binding FNaam}" FontWeight="Bold" Style="{StaticResource txbstyle}"  />
        <Image Source="{Binding Foto}" Grid.Column="1" />
    </Grid>
</DataTemplate>
```

Figuur 3.16: RelativeSource binding

3.5 TypeConverter

Zie TypeConverter

Jet Daels

Hoofdstuk 4

Delegates

4.1 Delegates: duiding

In een volledig object georiënteerde programmeeromgeving is het mogelijk om code- onderdelen als objecten te aanzien. Dit betekent dat deze onderdelen (routines, classes, assemblies, types, ..) instanties zijn van klasses die dergelijke code elementen beschrijven.

Routine- objecten (het onderwerp van dit hoofdstuk) zijn instanties van een *Delegate-type*. Deze Delegate- types zijn klasses die erven van de *Delegate*- class. Deze laatste is zelf geen Delgate-type (er zijn geen routines die instanties zijn van de basis Delegate-class).

In het hoofdstuk omtrent reflection zullen we ook andere code onderdelen als objecten verwerken. Routines zullen daar ook als instanties van (Reflection-) classes terugkeren. In een reflection context zullen deze instanties vooral beschrijvende (meta-) informatie bevatten.

Delegates spelen in .NET een belangrijke rol bij het implementeren van events, multithreading en codeplumbing. Ook de implementatie van WCF en (attached) dependency properties maken gebruik van delegates. Delegates zijn dan ook zonder twijfel een belangrijke werkstuk voor de betere programmeur!

4.1.1 Relevante namespace

- *System*

4.2 Delegates: technische intro

4.2.1 Demo-code en toelichting

De meest voor de hand liggende actie die we op een routine willen toepassen, is deze uitvoeren. Hierbij is het van belang te weten welke argumenten moeten voorzien worden en van welke type het eventuele resultaat een instantie zal zijn. Een delegate- type definitie zal zich hiertoe beperken: welke zijn de argumenttypes, en welk is het eventuele resultaat type? De naam van de routine en de parameters, en de geïmplementeerde logica, zijn volledig irrelevant voor de delegate type definitie. Een voorbeeld delegate- type definitie:

```
public delegate void DoeIetsMetTekstDelegate(string sTekst);
```

Figuur 4.1: Delegate definitie

Merk hierbij volgende punten op:

- het keyword *delegate* heeft aan dat we een delegate-type definiëren (een speciale klasse, zonder body);
- het return type wordt genoteerd voor de naam van het delegate type. In het voorbeeld is dit return type void (we verwachten geen resultaat);
- de argumentnamen zijn niet van belang (en moeten zeker niet overeenstemmen met de argumentnamen gebruikt in de routines die instanties van deze klasse zullen zijn). De argumenttypes zijn *wel* belangrijk!

```
public delegate void doeIetsMetTekst(String tekst);

public partial class MainWindow : Window
    {
    private void btnDemoUCLC_Click(object sender, RoutedEventArgs e)
        {
        doeIetsMetTekst routine = new doeIetsMetTekst(toonUC);  //maak een routinereferentie en koppel ze
        routine.Invoke("hAllO");    //start de methode gekoppeld aan deze routinereferentie, en voorzie de oproep van de nodige parameters
        routine = new doeIetsMetTekst(toonLC);
        routine.Invoke("hAllO");
        }
    private void toonUC(string tekst) { MessageBox.Show(tekst.ToUpper()); }
    private void toonLC(string tekst) { MessageBox.Show(tekst.ToLower()); }
```

Figuur 4.2: Delegate demo: uppercase en lowercase

Bovenstaande demo-code zal u wellicht niet overtuigen van het belang van delegates: het aanroepen van een methode wordt gewoon complexer gemaakt. Verder in dit hoofdstuk behandelen we een aantal (niet exhaustief (=uitputtend)) situaties waarbij het gebruik van delegates een elegante oplossing biedt.

Jef Daels

Een delegate- definitie is niet hetzelfde als een signatuur definitie: de routinenaam maakt onderdeel uit van de signatuur (en niet van het delegate- type), en het returntype is geen onderdeel van de signatuur, maar wel van belang in de delegate definitie.

4.2.2 Covariance en contravariance

Wat indien een routine hoofding niet volledig overeenstemt met de delegate definitie? Een routine kan op twee manieren afwijken van een delegate type definitie, en toch nog als instantie van dit delegate type worden aanzien: covariance en contravariance. Beide worden beschreven in Covariance and Contravariance in Delegates (C# Programming Guide). Een combinatie van beide is natuurlijk ook toegelaten.

4.2.2.1 Covariance

Indien het returntype van een routine een subclasse is van het return type van het delegate type, dan is deze routine toch een instantie van dit delegate type. Uiteraard levert dit geen problemen op, gezien elk resultaat kan gecast worden naar het return type van het delegate type.

4.2.2.2 Contravariance

Indien een argument van de routine minder gespecialisserd is dan het overeenkomstige argument in het delegate type is er uiteraard ook geen probleem: elke actuele parameter moet voldoen aan het delegate (formele) parameter type, en voldoet dus zeker ook aan het algemenere routine parameter type.

4.3 Delegates en events

Events leveren een mechamisme waarbij een server-object (levert diensten) initiatief neemt om code door een client-object te laten uitvoeren. In .NET wordt dit gerealiseerd door middel van delegates (dit in tegenstelling tot JAVA waar men gebruikt maakt van interfaces (denk bijvoorbeeld aan de *ActionListener* interface)).

Het client of server zijn van een object ten opzichte van elkaar hangt af van de te leveren dienst: indien we een Button op een Window plaatsen dan is deze Button client van het Window omwille van de visuele ondersteuning (het Window levert een dienst), en is het Window evenzeer client van de Button omdat de Button onder andere een click- dienst aanlevert. Het is dus zeker niet zo dat een client/server relatie gelezen moet worden als een klein/groot- relatie.

4.3.1 Definitie van een event

De definitie van een event voorziet een delegate- typed field van het keyword *event*:

```
public delegate void EventVoorbeeldNietVolgensGuidelines(string tekst); //delegate voor het event
public class DemoTekstVak : TextBox
    {
    public event EventVoorbeeldNietVolgensGuidelines TekstIsVijfLang; //eventdefinitie
    public DemoTekstVak()
        {
```

Figuur 4.3: Event definitie (zonder guidelines)

De delegate in de eventdefinitie beschrijft de client-routines die aan dit event gekoppeld kunnen worden. Bemerk dat de code in dit voorbeeld *niet* voldoet aan de guidelines omtrent eventhandling.

Bovenstaande code is onderdeel van een gespecialiseerde TextBox klasse die voorzien wordt van een TekstIsVijfLang event. Het is de bedoeling dit event af te vuren indien de tekst in het tekstvak exact 5 tekens lang wordt.

Jef Daels

4.3.2 Abonneren (to subscribe) op een event

De syntax nodig om een client te abonneren op een event is verschillend met deze van VB.NET (waarbij een *handles* clause achteraan een routine werd toegevoegd). In $C^{\#}$wordt een routine gekoppeld aan een event door middel van de *+=*- operator:

```
public MainWindow()
    {
    InitializeComponent();
    txt5Tekens.TekstIsVijfLang += new EventVoorbeeldNietVolgensGuidelines(txt5Tekens_TekstIsVijfLang);
    }
void txt5Tekens_TekstIsVijfLang(string tekst)
    {
    Console.WriteLine(tekst + " is 5 tekens lang");
    }
```

Figuur 4.4: Event abonnering

Visual Studio ondersteunt het abonneren op volgende wijze:

- indien u na de *+=*- operator de *TAB* toets indrukt wordt automatisch een correcte right-hand expressie gegenereerd;
- indien u hierna nog eens *TAB* indrukt zal ook een correcte eventhandler routine gegenereerd worden, rekening houdend met de event definiërende delegate.
- indien u dubbelklikt op een control wordt er een *+=*- expressie aangemaakt in de generated code en wordt een passende eventhandler in de *.cs*- fiel geplaatst;

4.3.2.1 Event subscription: verschil met VB.NET

- Handles- clause: Hiervoor bestaat geen $C^{\#}$- equivalent: $C^{\#}$kent enkel de *+=*- operator om routines aan events te koppelen;

- AddHandler: De VB.NET handles clause is niet in staat om routines *at run-time* aan events te koppelen. Hierdoor is deze werkwijze ongeschikt om events van bijvoorbeeld dynamisch toegevoegde controls op te vangen. VB.NET kent de *AddHandler* instructie om at run-time routines aan events te koppelen. Indien u de documentatie erop naleest merkt u dat VB.NET de *AddressOf*- operator gebruikt in combinatie met de AddHandler instructie.

4.3.3 Afvuren van een event

De klasse die het event definieert kan dit event ook uitvoeren. Het is belangrijk om te testen of er clients zijn voor dit event (indien er geen clients zijn is het field *null* en zou het afvuren een uitvoeringsfout veroorzaken):

```
public delegate void EventVoorbeeldNietVolgensGuidelines(string tekst); //delegate voor het event
public class DemoTekstVak : TextBox
    {
    public event EventVoorbeeldNietVolgensGuidelines TekstIsVijfLang; //eventdefinitie
    public DemoTekstVak()
        {
        this.TextChanged += new TextChangedEventHandler(DemoTekstVak_TextChanged);
        }

    void DemoTekstVak_TextChanged(object sender, TextChangedEventArgs e)
        {
        if (this.Text.Length == 5)
            if (TekstIsVijfLang != null)    //minstens 1 client voor dit event nodig, anders null
                TekstIsVijfLang(this.Text); //starten van het event (voor alle clients)
        }
    }
```

Figuur 4.5: Event afvuren (zonder guidelines)

Bemerk dat in bovenstaande code, die het afvuren van een event illustreert, er ook een event abonnering aanwezig is: het object abonneert zichzelf (in de constructor) op zijn eigen *TextChanged*- event (door middel van de *+=- operator*) om na te kijken of de tekst na wijziging exact vijf tekens lang is.

4.3.3.1 Afvuren van een event: verschil met VB.NET

- *RaiseEvent* is de instructie om in VB.NET een event af te vuren. Het is *niet* nodig om te controleren of er clients zijn voor dit event.

4.3.4 Meerdere clients voor hetzelfde event

Het is perfect mogelijk om meerdere event-handlers te koppelen aan eenzelfde event. Bij het afvuren van het event zullen de gekoppelde routines in volgorde, één voor één uitgevoerd worden.

4.4 Event Guidelines!

Hoewel er technisch geen beperkingen worden opgelegd aan de event definiërende delegates worden door Microsoft volgende guidelines beschreven:

- het return type is *void*;
- de delegate heeft exact twee argumenten:
  - Object sender: het object waardoor het event wordt gestart;
  - EventArgs e: EventArgs mag vervangen worden door een klasse die overerft van EventArgs indien u extra informatie aan het event wenst toe te voegen (voorbeelden hiervan zijn de mouselistener events);
- het afvuren van het event gebeurt in een overschrijfbare (*protected virtual*) methode. Dit laat ervende klasses toe om het event gedrag aan te passen (zie DemoButton in het demo project). De methodenaam is *onEvent* waarbij Event natuurlijk wordt vervangen door de naam van het event.
- gebruik *System.EventHandler<T>* in plaats van zelf aangemaakte delegate definities.

Deze guidelines *moeten* in deze cursus worden toegepast opdat uw event-implementaties als correct zouden worden aanzien.

Als een eerste oefening kan men de event implementatie, zoals hierboven beschreven, hercoderen zodat ze voldoet aan de guidelines.

Wie niet (voldoende) bekend is met *generics* (*System.EventHandler<T>*) wordt verondersteld dit op te zoeken. Indien zelfstudie onvoldoende duidelijkheid schept kan u vragen stellen in de komende lessen.

Jef Dedls

4.5 Omtrent Events:

4.5.1 Static events

Meestal associëren we een event met een instantie. Om te abonneren op een event van een instantie hebben we een referentie naar die instantie nodig. Dit is niet altijd mogelijk: veronderstel dat client code wenst te reageren op het aanmaken van een nieuw object van een klasse: het is niet mogelijk in te tekenen op het *PersoonCreated* event van een nog onbestaand object. .NET voorziet de mogelijkheid om static events te definiëren: de client abonneert dan op een (static) event van een klasse:

```
public class clsPersoon
    {
    public static event EventHandler<PersoonCreatedEventArgs> PersoonCreated;
    public string FNaam { get; set; }   //property definitie
    public string VNaam { get; set; }   //property definitie
    public clsPersoon() : this("","")
        {
        }
    public clsPersoon(string fnaam, string vnaam)
        {
        FNaam = fnaam;
        VNaam = vnaam;
        if (PersoonCreated != null)
            PersoonCreated(null, new PersoonCreatedEventArgs(this));
        }
    public override string ToString()
        {
        return this.FNaam + ", " + this.VNaam;
        }
    }
public class PersoonCreatedEventArgs : EventArgs
    {
    private clsPersoon _persoon;
    public PersoonCreatedEventArgs(clsPersoon persoon) { _persoon = persoon; }
    public clsPersoon persoon { get { return _persoon; } }
    }
```

Figuur 4.6: Static event definitie (volgens de guidelines)

```
public MainWindow()
    {
    InitializeComponent();
    clsPersoon.PersoonCreated += new EventHandler<PersoonCreatedEventArgs>(clsPersoon_PersoonCreated);
    }

void clsPersoon_PersoonCreated(object sender, PersoonCreatedEventArgs e)
    {
    Console.WriteLine(e.persoon.ToString());
    }
```

Figuur 4.7: Static event client

4.5.2 Delegates of Interfaces: .NET of JAVA events

In .NET gebruikt men delegates om eventhandling te realiseren. In JAVA definieert men interfaces om eventhandling mogelijk te maken: een instantie van een interface implementerende klasse kan dan als eventhandler aan een object toegekend worden. Welke is te verkiezen?

Bij manuele codering (wie doet dat nog?) moet in JAVA elk interface element voorzien worden, ook al bent u enkel geïnteresseerd in een specifieke routine. In .NET moet u enkel die events die u interesseren van code voorzien. Beide oplossingen zijn voldoende eenvoudig om een intuitief gebruik mogelijk te maken (zeker indien de IDE u ook nog wat helpt). Beide benaderingen zijn dan ook sterk gelijkwaardig: ik verkies er geeneen boven de andere.

4.5.3 Routed events

Voorgaande eventhandling is reeds van bij de start van .NET aanwezig in het framework. Recentelijk (WPF, .NET 3.5) werden ook *routed events* geïntroduceerd. Deze events zijn WPF specifiek en worden geïntroduceerd om volgende problemen op te lossen:

- veronderstel een Button, met daarop een tekening (Image);
- de gebruiker klikt op de tekening: wellicht wil hij de button inklikken? Misschien ook niet? Hoe wordt de Button op de hoogte gesteld van het klikken op de Image? Wat indien er geen Image is?

Een routed event is een event (anders dan voorgaande) om binnen een geneste lijst van controls een gebeurtenis door te geven. We spreken van *tunneling events* indien de container het event doorgeeft naar zijn child, en van *bubbling events* indien het child het event doorgeeft aan zijn container. Een parent (bijvoorbeeld Window)is in staat om te subscriben (addHandler) op het routed event (bijvoorbeeld *TextChanged*- event van een bij abonnering niet gekend child (zie ook Sudoku routed events).

Indien een WPF- gebeurtenis zich voordoet zal het doorgeven van events in twee stappen gebeuren:

- eerst wordt het event van de containers naar de children doorgegeven: *tunneling*. Deze events worden prefixed met *preview*;
- daarna volgt een bubbling fase waarbij de gewone event naam wordt gebruikt;
- indien een control meent het event correct af te handelen kan de handled property van het tweede event-argument op *true* worden geplaatst, wat het doorgeven van die specifieke fase (grotendeels) stopt.

Interessant om weten:

- routed events bestaan enkel in een WPF-omgeving. Ze kunnen dan ook niet gebruikt worden om eventhandling buiten de WPF omgeving (uw visualisatie) te implementeren.
- snoop is een leuke tool om WPF-constructies te onderzoeken. Ook het verloop van routed events kan met deze tool worden nageplozen.

4.6 Delegates en multithreading

4.6.1 Multithreading: wat bedoelt u?

Bij het uitvoeren van een programma worden instructies één voor één door de CPU verwerkt. Een thread is een onderdeel van een proces (programma) dat in staat is om opeenvolgende instructies uit te voeren. Een multithreaded programma heeft meerdere onafhankelijk van elkaar uitvoerende threads. Wanneer het programma wordt uitgevoerd door een multithreaded operating system worden verschillende zaken op hetzelfde moment uitgevoerd.

4.6.2 Multithreading: iets voor ons?

Multithreaded programmeren is *ongeloofelijk* delicaat: indien gelijktijdige threads eenzelfde variabele wensen aan te passen krijgen we de problemen die we kennen vanuit de cursus databases: *lost update*, *inconsistent read* en *ghost data* om er slechts enkele te noemen. Het schrijven van correcte multithreaded toepassigen is dan ook geen *walk in the park/piece of cake*.

Betekent dit dat we geen multithreading kunnen programmeren? Natuurlijk niet: het ophalen van een grote lijst gegevens, het verwerken van een lijst gegevens die niet gewijzigd worden: allemaal ideale kandidaten om multithreaded uitgewerkt te worden. Indien er zich concurrency (gelijktijdigheid) problemen kunnen voordoen zullen we locks gebruiken. Bij het database programmeren worden locks door het RDBMS geplaatst. In $C^{\#}$(en VB.NET) kunnen we dit zelf programmeren met behulp van de *lock*- instructie.

Wanneer we spreken over multithreaded toepassingen hebben we het veelal ook over:

- parallel processing: parallelle (gelijktijdige) uitvoeringen van meerdere threads impliceren multithreading;
- asynchrone verwerking (in tegenstelling tot synchrone verwerking bij single threaded toepassingen): hierbij wordt een routine gestart en wachten we niet op het einde van deze routine vooraleer we verder gaan. Asynchrone verwerking wordt in .NET gerealiseerd met behulp van delegates. Een voor de hand liggend probleem dat moet worden opgelost: hoe weet mijn programma dat een asynchrone uitvoering afgelopen is (om bijvoorbeeld een *printen beëindigd* boodschap te kunnen tonen?

Multithreaded applicaties zijn ook op een single- core CPU belangrijk: zo kan men het printen van een document op de achtergrond laten uitvoeren, terwijl de user-responsiveness niet daalt. Met de introductie van multi-core CPU's wordt het ontwikkelen van multi-threaded toepassingen nog interessanter: een single-threaded toepassing zal op een quad core nooit meer dan vijventwintig procent van de beschikbare rekenkracht kunnen gebruiken. Dit maakt het multi-threaded zijn van uw toepassing belangrijk indien u het volle potentieel van de machine wenst te gebruiken.

4.6.3 Asynchrone processing en delegates

4.6.3.1 Asynchroon, geen AsyncCallback

Bij een synchrone verwerking zal een routine de volledige utivoering van een instructie (eventueel een deelprobleem) afwachten vooraleer de volgende instructie wordt gestart. Zo zal de uitvoering van btnDemoUCLC_click in Delegate demo: uppercase en lowercase na het tonen van de eerste messagebox wachten tot de gebruiker op *ok* klikt vooraleer de tweede messagebox wordt getoond.

Wanneer we de code aanpassen door de *invoke/1-* methods van de delegates te vervangen door *BeginInvoke/3* dan worden de delegate routines asynchroon ten opzichte van de oproepende methode uitgevoerd:

```
private void btnAsyncDemo_Click(object sender, RoutedEventArgs e)
    {
    doeIetsMetTekst routine = new doeIetsMetTekst(toonUC);  //maak een routinereferentie en koppel ze
    routine.BeginInvoke("hAllO", null, null);    //start de delegate asynchroon
    routine = new doeIetsMetTekst(toonLC);
    routine.BeginInvoke("hAllO",null,null);
    }
```

Jef Daels

Figuur 4.8: Asynchrone verwerking demo

Op te merken bij de nieuwe situatie:

- deze uitvoering resulteert in drie threads: de thread die het klik- event afhandelt, en een tread voor elk van de asynchrone oproepen;

- *BeginInvoke/3* kent drie parameters in plaats van één: de twee laatste parameters van een asynchrone delegate oproep bevaten een callback routine (de routine die wordt uitgevoerd als de asynchroon gestartte routine eindigt) en een status object als laatste argument (beide komen in het volgende punt aan bod). Het is de bedoeling om in dit status object informatie, verzameld gedurende de asynchrone uitvoering (bijvoorbeeld het goed of slecht aflopen) terug te geven aan de oproeper. Dit laatste argument is (later) terug te vinden als state property van de *IAsyncResult-* parameter van de *AsyncCallback* routine.

- de callback methode wordt via een AsyncCallBack- delegate aan de asynchrone oproep meegegeven (de callback methode is die routine die zal worden uitgevoerd wanneer de asynchrone oproep afgelopen is). Het is belangrijk op te merken dat de callback routine wordt uitgevoerd op de asynchrone thread!

4.6.3.2 Asynchroon, met AsyncCallback, zonder UI

```
private void btnAsyncDemo_Click(object sender, RoutedEventArgs e)
    {
    doeIetsMetTekst routine = new doeIetsMetTekst(toonUC);  //maak een routinereferentie en koppel ze
    routine.BeginInvoke("hAllO", new AsyncCallback(asyncAfgelopen), this);    //start de delegate asynchroon
    routine = new doeIetsMetTekst(toonLC);
    routine.BeginInvoke("hAllO",null, null);
    }
private void asyncAfgelopen(IAsyncResult iar)
    {
    toonResultaat(iar);
    }
private void toonResultaat(IAsyncResult iar)
    {
    object state = iar.AsyncState;  //bevat laatste argument van de oproep
    this.Title = "afgelopen";
    }
```

Figuur 4.9: AsyncCallback demo

In bovenstaande figuur zal de eerste asynchrone uitvoering bij het aflopen de routine *asyncAfgelopen* uitvoeren (op de nieuwe thread). In het *IAsyncResult* argument vinden we informatie omtrent de uitgevoerde routine. De callback routine wordt uitgevoerd op de nieuw gestarte thread. Bovenstaande stukje code loopt fout omdat we op de nieuwe thread de UI wensen aan te passen:

Figuur 4.10: UI-thread problem

In een Windows-omgeving is het enkel mogelijk om de UI-elementen aan te passen op de thread die deze UI-elementen aanmaakte. Indien we in de UI feedback omtrent een afgelopen asynchrone methode wensen te zien, zullen we in staat moeten zijn om naar de UI-thread terug te keren. U raadt het al: dit gebeurt met behulp van delegates.

4.6.3.3 Asynchroon, met AsyncCallback, met UI

```
private void btnAsyncDemo_Click(object sender, RoutedEventArgs e)
    {
    doeIetsMetTekst routine = new doeIetsMetTekst(toonUC);  //maak een routinereferentie en koppel ze
    routine.BeginInvoke("hAllO", new AsyncCallback(asyncAfgelopen), this);    //start de delegate asynchroon
    routine = new doeIetsMetTekst(toonLC);
    routine.BeginInvoke("hAllO",null, null);
    }
private void asyncAfgelopen(IAsyncResult iar)
    {
    Dispatcher.BeginInvoke(new AsyncCallback(toonResultaat),iar);   //Terug naar de ui thread
    }
private void toonResultaat(IAsyncResult iar)
    {
    object state = iar.AsyncState;  //bevat laatste argument van de oproep
    this.Title = "afgelopen";
    }
```

Figuur 4.11: Dispatcher demo

Het terugkeren naar de UI-thread gebeurt afhankelijk van de UI-technologie:

- WPF: elke visueel element heeft een *Dispatcher* property. Deze kan een thread starten op zijn UI-thread door middel van de *BeginInvoke/2* methode. Hoewel deze methode dezelfde naam draagt als deze die de delegate routines asynchroon start is het een andere methode (geen twee extra argumenten bijvoorbeeld);
- WinForms: elke control heeft een *BeginInvoke* methode. Deze start een delegate op de thread die de control maakte (de UI-thread).

Het is interessant op te merken dat een *Messagebox.show* oproep zichzelf terug op de gepaste thread plaatst.

4.6.4 Multithreading met de Thread klasse

Een alternatief om multithreaded toepassingen te maken kan gebruik maken van de *Thread*-klasse. Dit valt buiten de scope van deze les.

Jeff Maels

4.6.5 Event based Ansynchronous Pattern (EAP)

Het Event based Asynchronous Pattern plaatst het asynchroon ontwikkelen in de alomgekende event- context:

- zowel de langlopende taak (op de andere thread) als de callback worden gekoppeld aan events (zie het BackgroundWorker voorbeeld hieronder);
- op basis van de uitleg omtrent events zou het duidelijk moeten zijn dat dit een wrapper-oplossing rond de asynchroon uit te voeren delegate en zijn callback is.
- het pattern voorziet ook de mogelijkheid om een langdurende bewerking te onderbreken en om voortgangs events af te vuren. U vindt dit terug in de documentatie :).

De BackgroundWorker class is een mooi voorbeeld van deze benadering:

- het DoWork- event: ik kan het niet beter uitleggen dan wat u in MSDN vindt. Bemerk de relatie met RunWorkerAsync
- het RunWorkerCompleted- event wordt ook netjes uitgelegd op MSDN.

4.6.6 Asynchroon programmeren made easy

Voorgaande code, waarbij het nodig was om *Callback delegates* en *DispatcherThreads* te begrijpen maakte het niet eenvoudig om multithreaded toepassingen op te zetten. In .NET 4.5 werd gepoogd om dit (syntactisch) een flink stuk te vereenvoudigen met de introductie van de *await*- en *async* keywords:

```
public async Task XMLSerialise()
    {
    //dit wordt enkel gebruikt als er nog geen bord geserialiseerd was (su
    StorageFolder storageFolder = ApplicationData.Current.LocalFolder; //a
    StorageFile sf = await storageFolder.CreateFileAsync("numberlegionsspl
    Stream s = await sf.OpenStreamForWriteAsync();
    XmlSerializer serializer = new XmlSerializer(typeof(SplashPage));

    serializer.Serialize(s, this);
    s.Dispose();
    }
```

Figuur 4.12: Await- keyword

- het *await*- keyword impliceert volgende zaken:
  - de oproep rechts van het *await*- keyword wordt asynchroon uitgevoerd;
  - de rest van de methode wordt slechts uitgevoerd nadat de asynchrone oproep eindigt. De compiler zal hiertoe de rest van de methode in een delegate wrappen en als callback met de asynchrone oproep meegeven;

- het *async*- keyword in een routine hoofding heeft aan dat deze routine *awaited* kan opgeroepen worden. Meer nog, indien een routine een *awaited*- oproep bevat (typisch een methode oproep die lang kan duren) moet deze routine verplicht *async* gedefinieerd worden. Dit maakt het mogelijk (niet noodzakelijk) om deze routine zelf ook terug *awaited* op te roepen;
  - indien u een *async* methode *niet* awaited oproept dan wordt ze synchroon uitgevoerd (en blokkeert ze dus de oproepende thread).
  - async methodes moeten een return type *Task* (indien geen resultaat- waarde) of *Task<resultaattype>* indien een resultaat wordt terug gegeven.
- bij het ontwikkelen van *Modern Style* apps (Microsoft tablet toepassingen) bent u verplicht deze keywords te gebruiken: het manipuleren van files, het raadplegen van webservices: al deze functionaliteit is enkel nog in een asynchrone versie beschikbaar;
- wat u momenteel nog ontbreekt is de kennis om zelf een asyn methode te ontwikkelen (zonder gebruik te maken van een andere async methode die u await). Dit zal aan bod komen in het hoofdstuk omtrent backtracking waar we onze artificiële intelligentie asynchroon zullen oproepen.
  - voor wie hierop niet kan wachten: we zullen gebruik maken van de TaskCompletionSource klasse.

Jef Daels

4.6.7 CancellationToken

De CancellationToken structure wordt gebruikt in GeolocationCS sample. Dit token dat bij het starten van een Task wordt meegegeven geeft de caller de mogelijkheid om de Task af te breken. Hiertoe is het nodig dat de gestarte Task regelmatig dit token raadpleegt en indien nodig zijn bewerkingen afbreekt.

4.6.8 Thread.Sleep

Indien meerdere threads met dezelfde prioriteit uitgevoerd moeten worden kan het interessant zijn om de uitvoerende thread te onderbreken zodat de andere thread met dezelfde prioriteit aan bod kan komen. Hiertoe kan u gebruik maken van de (statische) Thread.Sleep - methode. U zal dit nodig hebben om de multithreaded teller oefening op te lossen.

4.6.9 Meer info?

Over multithreading zijn boeken, wellicht ganse bibliotheken, geschreven. De laatste jaren is er hernieuwde aandacht voor deze topic omwille van de multi core processoren (de programmeer talen worden aangepast om parallelle verwerking gemakkelijker te ondersteunen). Een eventueel startpunt om zich in deze materie te verdiepen kan je vinden op Asynchronous Programming Overview. Deze link is wellicht niet beter dan uw eigen favoriete instap link.

4.7 Code plumbing, aka 'the strategy pattern'

Deze laatste topic behandelt het 'at run time' vervangen van code in *één welbepaald* object. Dit is helemaal anders dan overerving: bij overerving zullen we code at compile time aanpassen voor *elke* instantie van de ervende klasse.

4.7.1 Codeplumbing: een XMLViewer

Veronderstel dat u betrokken bent bij de ontwikkeling van een algemene XML-viewer. Deze is in staat om elke mogelijke XML-file in een treeview te visualiseren. Omdat u op voorhand niet alle XML-tags van uw potentële klanten kent kan u in uw XMLviewer geen tag- afhankelijke manipulaties coderen: u weet niet op voorhand hoe een XML-file met CD-gegevens vertoont moet worden. Het is onze doelstelling een XMLViewer control te ontwikkelen waarin de gebruiker bepaalde routines kan vervangen door zijn eigen implementaties. Wij voorzien default implementaties die een basis functionaliteit aanbieden zodat de XML-viewer op zijn minst een standaard gedrag vertoont. We gaan hiervoor als volgt te werk:

- maak een XMLViewer usercontrol die erft van TreeView. Deze overloopt bij het verwerken van een XML-file recursief de verschillende tags en toont deze. Wellicht is het interessant om ook een eigen TreeViewItem klasse te maken. Houd rekening met volgende elementen:

  - plaats de berekening van de getoonde tekst voor elke XML-node in een aparte routine (met de XML-node als argument);
  - plaats de berekening van de kleur voor elke XML-node in een aparte routine (met de XML-node als argument);
  - plaats de test die bepaalt of subnodes moeten verwerkt worden in een aparte routine (met de XML-node als argument);

  U merkt dat elke functionaliteit die we later voor de gebruiker aanpasbaar willen stellen in aparte routines terechtkomen. Deze routines implementeren het default gedrag (en mogen enkel gebruik maken van de parameters, niet van de module variabelen);

- voorzie voor elke default routine een delegate- class waar deze toe behoort;

- definieer voor elke default routine een field van het overeenkomstige delegate type, en koppel dit field in de constructor aan zijn default implementatie;

- vervang elke oproep van de default-routines door een *invoke*- aanroep van het field. Dit start de methode in het field waarin default onze eigen routines te vinden zijn.

- voorzie *setters* voor elk delegate- field in de klasse. Dit geeft gebruikers van onze XMLViewer de kans om onze default methodes door zijn eigen routines te vervangen.

Figuur 4.13: XMLViewer en code plumbing

De linker versie toont het default gedrag van de XMLViewer. De rechter versie toont dezeldfe informatie in dezelfde uitvoering nadat de XMLViewer control werd voorzien van aangepaste visualisatie routines.

De screenshot toont een WPF- programma. De techniek op zich is uiteraard ook bruikbaar in een WinForms context.

4.7.2 Codeplumbing pitfalls

Wanneer we codeplumbing toepassen verdienen volgende punten uw aandacht:

- vermijd delegate properties indien u usercontrols maakt. Een property wordt door Visual Studio in het property window getoond (door de property te serialiseren). In het geval van een delegate zal dit in Visual Studio voor problemen zorgen (herhalende foutboodschappen, ook na het afsluiten en heropenen van het project in Visual Studio).
- laat uw codeplumbing routines *enkel* gebruik maken van de routine parameters. De delegate wordt immers uitgevoerd waar hij werd gedefinieerd. De module niveau variabelen zijn dus soms wel (wanneer de default routines worden uitgevoerd), soms niet (wanneer de nieuwe variant wordt uitgevoerd) deze van het object dat de delegate oproept.

4.8 Delegates: faits divers

4.8.1 Action en Func

Omdat in veel situaties gelijkaardige delegate definities nodig zijn werden in .NET een aantal delegate definities voorzien. Het is aan te raden deze indien mogelijk te gebruiken, eerder dan zelf nieuwe delegate klassen te definiëren;

4.8.1.1 Action

De klasse Action is een delegate klass voor routines zonder argumenten en zonder resultaat. Action$< T >$ is een (generische) klasse voor routines met exact één argument (gelijk welk type) en zonder resultaat.

4.8.1.2 Func

De generische klasse $Func< outTResult >$ wordt gebruikt voor functies zonder argumenten (en gelijk welk resultaat type), en $Func< inT, outTResult >$ wordt gebruikt voor functies met exact één argument (willekeurig type) en een willekeurig resultaat type.

4.8.2 Meer dan (begin)-invoke

Delegates ondersteunen meer dan enkel een (begin)invoke routine. Zo bestaat ook de mogelijkheid om verschillende delegates (van hetzelfde delegate type) te combineren (geen co- of contravariance mogelijk) en betaan er ook multicast delegates (die achter de schermen gebruikt worden om meerdere clients toe te laten op hetzelfde event te abonneren)

4.8.3 Timers

.NET ondersteunt verschillende Timer- classes (zie ook Comparing the Timer Classes in the .NET Framework Class Library). Indien u deze tekst doorneemt zal u begrijpen dat niet elke Timer 'tikt' op de UI-thread, wat betekent dat niet elke Timer geschikt is om UI-wijzigingen rechtstreeks aan te sturen (zie ook Asynchroon, met AsyncCallback, zonder UI).

Bemerk trouwens dat de constructor van de *System.Threading.Timer* klasse onder andere een delegate object verwacht. De twee andere timer-objecten werken met events, wat gezien Delegates en events eigenlijk niet echt een verschil uitmaakt.

4.8.4 Asynchrone eventverwerking

Wie de voorgaande topics goed begrepen heeft zou in staat moeten zijn om een eigen, asynchrone eventimplementatie op te zetten: een uitstekende oefening voor thuis.

4.8.5 Anonymous methods (.NET 2.0)

Bij het maken van een delegate instantie werd in voorgaande tekst telkens gebruikt gemaakt van een methode met gepaste signatuur. Indien men deze extra methode definitie wenst te vermijden (bijvoorbeeld bij het definiëren van MVVMCommands) kan men gebruik maken van *anonieme methodes*. Deze bestaan uit een (geschikte) formele parameterlijst definitie en een body:

```
btnAnonymousMethod.Click +=
    delegate(object o, RoutedEventArgs e)
        {MessageBox.Show("test1"); MessageBox.Show("test2");}
    ;
```

Figuur 4.14: Anonymous method voorbeeld

Jef Daels

4.8.6 Lambda expressions (.NET 3.0)

Lambda expressions kunnen onder andere gebruikt worden om delegate objecten te creeëren. Ze bestaan uit twee stukken gescheiden door middel van de lambda operator =>. Links bevindt zich de input (het equivalent van de formele parameterlijst van een functiehoofding), rechts bevindt zich de body van de delegate. Het resultaat van de lambda expressie is (in dit geval) een delegate object:

```
btnAnonymousMethod.Click +=
    (object o, RoutedEventArgs e) =>
        {MessageBox.Show("test3"); MessageBox.Show("test4");
    ;
```

Figuur 4.15: Lambda expressie voorbeeld

Lambda expressions worden veel gebruikt in een LINQ-context, waar het injecteren van bijvoorbeeld selectielogica in een foreach-lus frequent voorkomt.

4.9 Delegates: niet voor elk programma een geschikte oplossing

- niet elk programma heeft nood aan delegates;
- (algemener:) niet elke 3nMCT-techniek is noodzakelijk om een goed programma te maken;
- (advies:) indien uw programma geen nood heeft aan delegates, gebruik ze dan niet: er zijn wellicht meer programma's te ontwikkelen zonder delegate- nood dan met.

4.10 Delegates: opgaves

- events: maak een eigen TextBox klasse met een extra event (guidelines!): dit event wordt afgevuurd indien het tekstvak leeg wordt. Demonstreer uw oplossing.
- multithreading: maak een window waarin verschillende tellers asynchroon kunnen oplopen. De teller wordt gerealiseerd door een control die erft van TextBox. Om dit te realiseren zal u de prioriteit van de visualisatie op background priority moeten uitvoeren. De demo toont de volgende elementen:
  - singlethreaded is er maar één activiteit actief;
  - multithreaded zijn er verschillende acties terzelfdertijd mogelijk: het tellen in twee textboxen en het verplaatsen van het window.
- maak een XMLViewer waarin de klant eigen routines kan inpluggen om specifieke xml's anders te visualiseren (zie ook XMLViewer). Demonstreer uw oplossing.

Jef Daels

Hoofdstuk 5

Reflection

5.1 Reflection: duiding

Reflection is technologie waardoor (meta-) informatie omtrent programma-elementen programmatorisch toegankelijk wordt voor eigen programma's. Deze programma-elementen bevinden zich veelal in *assemblies* die niet tot het uitvoerende programma zelf behoren (externe modules). Op basis van de bekomen informatie zal ons eigen programma wellicht extra functionaliteit aanbieden. In dit hoofdstuk zullen we reflection gebruiken om een *plugin* infrastructuur op te zetten.

Hoewel reflection, net als delegates, de mogelijkheid biedt om routines te starten zou het verkeerd zijn om delegates als onderdeel van reflection te interpreteren (of omgekeerd). Een belangrijk verschil is mijns insziens de op voorhand gekende interface elementen van een delegate- instantie (*invoke*, ..) terwijl reflection net de mogelijkheid biedt om met at-compile-time ongekende klasses en/of routines te werken.

Het belang van reflection kan moeilijk overschat worden. Visual Studio illustreert mooi de mogelijkheden van reflection: intellisense, het property-window, de compiler .. : veel van de programmeur ondersteunende features maken gebruik van reflection, en in het bijzonder van attributen. Ook het hoofdstuk omtrent Aspect Oriented Programming steunt sterk op reflection.

Tot slot zal de in dit hoofdstuk opgedane kennis toelaten om object oriëntatie als een syntactische constructie bovenop procedureel programmeren te zien: een inzicht dat hopelijk de blik verruimt.

5.1.1 Relevante namespaces

- *System.ComponentModel*
- *System.Reflection*

5.2 Reflection: attributen voorbeeld

Een programmeur kan code elementen voorzien van attributen om extra informatie aan het code element te koppelen. Deze attributen kunnen door andere programma's gelezen worden om een gepaste verwerking van het element mogelijk te maken. Deze verwerking staat los van de werking van het code element zelf die wellicht niet gewijzigd wordt. De namespace *System.ComponentModel* bevat attributen die door Visual Studio gebruikt worden. Een aantal van deze zijn enkel van toepassing in een WinForms context, en niet in WPF:

Figuur 5.1: ComponentModel attributen ($C^{\#}$)

Voor bovenstaande voorbeeld kunnen we volgende elementen vermelden:

- *DefaulEvent* en *DefaultProperty* worden enkel toegepast op klasse (usercontrol) niveau. Ze worden door Visual Studio gebruikt om in de designer volgende acties te ondernemen:
  - *DefaultProperty*: deze property wordt in het property window automatisch geslecteerd wanneer een control van deze klasse in de (Visual Studio-) designer wordt geselecteerd;
  - *DefaultEvent*: dit event wordt automatisch geselecteerd in het property window (event tab) wanneer een control van deze klasse in de (Visual Studio) designer wordt geselecteerd. Dubbel klikken op deze control zal code genereren om dit event op te vangen.
- *Category* is een attribuut waarmee de category (gebruikt in het propertywindow) van een property wordt opgegeven. Deze wordt door Visual Studio gebruikt (zie figuur);
- *Description* en *DefaultValue* zijn attributen die enkel in een Winforms Visual Studio designer omgeving gebruikt worden (de WPF designer in Visual Studio negeert deze attributen).

5.3 Reflection in .NET: meer dan attributen

5.3.1 Reflection in .NET: start

Maak een nieuwe solution met hierin een WPF-project. Maak een tweede(!) WPF project met volgende basis functionaliteit:

- voorzie een klasse *clsPersoon* met een aantal features;
- voorzie een knop op het default window om een nieuwe persoon aan te maken.

Compileer het nieuwe project en programmeer onderstaande code in het eerste project (zonder een referentie tussen beide projecten te leggen):

```
private void verwerkDemoUI()
    {
    Microsoft.Win32.OpenFileDialog ofd = new Microsoft.Win32.OpenFileDialog();
    ofd.Filter = ".NET assemblies|*.exe;*.dll";
    ofd.InitialDirectory = System.IO.Path.GetDirectoryName(
        System.Reflection.Assembly.GetExecutingAssembly().CodeBase);

    if (ofd.ShowDialog().Value)
        verwerkDemoUI(ofd.FileName);
    }
private void verwerkDemoUI(string sAssembly)
    {
    Assembly ass = Assembly.LoadFile(sAssembly);
    foreach (Type t in ass.GetTypes())
        Console.WriteLine(t.Name);
    }
```

Figuur 5.2: Assembly reflection ($C^{\#}$)

Indien we bij uitvoering van deze code en het tonen van de *OpenFileDialog* het compilatie resultaat van het tweede project selecteren, zal het uitvoerende programma in de console de naam van elk type in de gekozen assembly tonen: *MainWindow*, *App*, *Resources*, *Settings* en *clsPersoon*. Eenvoudig nazicht in Visual Studio leert dat dit klopt.

Bovenstaande code is een reflection startpunt: vertrekkend vanuit een assembly wordt informatie omtrent de code in de assembly opgevraagd (bijvoorbeeld de namen van de klasses). De Type-klasse ondersteunt flink wat getters: zo kunnen de attributen, constructoren, methodes, properties, events, ... worden opgevraagd. Elk van deze code elementen is terug ondervraagbaar. Het wordt sterk aangeraden om een aantal getters zelf te verwerken, ook al vind u hiervan geen afbeeldingen in deze tekst terug.

5.3.2 Hoe een instantie maken van een klasse in een assembly?

Normaal gezien zullen we, voor een gekende klasse clsX, volgende $C^{\#}$- syntax gebruiken:

```
clsX x = new clsX(..);
```

Figuur 5.3: Strongly typed instantiatie

In bovenstaande code wordt de gekende klasse clsX op twee plaatsen gebruikt: als type van de referentie en als type waarvan de constructor wordt opgeroepen. Indien we een type bekomen via reflection is bovenstaande onmogelijk: gezien het nodige type at compile time niet gekend is (clsX wordt gevonden in een at run-time geladen assembly) is bovenstaande lijn code onmogelijk (in $C^{\#}$), en moeten we twee problemen oplossen:

- het referentie type zullen we vervangen door *System.Object*. Omdat elke klasse (al dan niet indirect) erft van *System.Object* is elk instantie van een klasse te koppelen aan een referentie van dit root- type (polymorfisme door overerving). Natuurlijk kunnen we, indien we extra kennis omtrent de klasses hebben, eventueel een meer gespecialiseerd type gebruiken (zie het voorbeeld hieronder);
- het oproepen van de constructor van een klasse zal gecodeerd worden via de *Activator*-klasse:

```
private void verwerkIndienWindow(Type t)
    {
    if (t.Name.Equals("MainWindow"))    //dergelijke test vermijden => attributes
        {
        Window wdw = (Window) Activator.CreateInstance(t);
        wdw.Show();
        }
    }
```

Figuur 5.4: Activator.CreateInstance ($C^{\#}$)

- indien de constructor argumenten vereist kunnen deze aan de *CreateInstance*- oproep worden meegegeven (u bent *niet* beperkt tot parameterloze constructoren);
- omdat we weten dat een klasse met deze naam (beslissingen op basis van een klasse naam zijn te vermijden!) een Window is gebruiken we een *Window* referentie waarvan de *Show*-method *at compile time* gekend is.

Jef Daels

Een iets gecompliceerder voorbeeld illustreert het starten van een methode vertrekkend vanuit zijn object referentie:

```
private void verwerkIndienPersoon(Type t)
    {
    if (t.Name.Equals("clsPersoon"))  //dergelijke test vermijden => attributen
        {
        object p = Activator.CreateInstance(t);
        t.InvokeMember("FNaam", BindingFlags.SetProperty, null, p, new object[] { "Familie" });
        t.InvokeMember("VNaam", BindingFlags.SetProperty, null, p, new object[] { "lid" });

        string sPersoon = (string) t.InvokeMember("ToString", BindingFlags.InvokeMethod, null, p, new object[] { });
        Console.WriteLine(sPersoon);
        }
```

Figuur 5.5: Activator.InvokeMember ($C^{\#}$)

- de Type- klasse ondersteunt een *InvokeMember* methode waarmee een methode kan gestart worden. Het object dat de methode uitvoert is een argument van deze oproep, de nodige argumenten worden als een array van objecten voorzien;
- deze manier van werken is case sensitief!
- deze voorbeeld code is weinig representatief: de kans is klein dat de klasse- naam op voorhand gekend is, net zomin als men mag verwachten de methodes op voorhand te kennen. Indien de methodes op voorhand gekend zijn is het wellicht aangewezen om met een interface te werken. Op die manier is een verwerking als deze van het window (zie vorige voorbeeld) mogelijk.

5.3.3 .NET en attributes

5.3.3.1 .NET en attribute informatie opvragen

De klasse *Type* ondersteunt een *GetCustomAttributes* methode waarmee de (custom-) attributen van een klasse kunnen opgevraagd worden. Ook andere code elementen hebben een *GetCustomAttributes* property. Op basis van de extra informatie gevonden in een attribuut kan een gebruikers programma gestuurd worden. Verschillende attribute klasses ondersteunen natuurlijk verschillende properties. Een voorbeeld attribuut zullen we uitwerken in onze eigen plugin infrastructuur implementatie.

5.3.3.2 Hoe zelf Attribute klasses maken in .NET?

Een goede beschrijving kan u vinden op Writing Custom Attributes. Als opmerking hierbij zou ik formuleren dat ik verwacht dat de meeste properties *read-only* gedefinieerd zijn (en niet *read-write* zoals in de voorbeeldcode).

Er wordt zeker verwacht dat u in staat bent om attributen te definiëren die enkel op classes of enkel op methodes toegepast kunnen worden.

5.4 Een eigen plugin- infrastructuur!

5.4.1 Een eigen plugin- infrastructuur: vereisten

Ik wens volgende situatie te realiseren:

- we moeten de mogelijkheid voorzien om in een toepassing extra logica uit een op voorhand niet gekende *assembly* te laden;
- de extra logica wordt aangeboden door middel van plugins. Een plugin- klasse erft van *Window*, en alle relevante extra's worden ontsloten via deze klasses. Er kunnen Window klasses in de assembly worden gevonden die geen plugins zijn (maar bijvoorbeeld gestart kunnen worden door een plugin in dezelfde assembly);
- het is de bedoeling om bij het laden van een assembly alle gevonden plugins via het menu te presenteren. De menu-tekst voor een plugin moet ook in de assembly gevonden worden;
- zie demo.

5.4.2 Een eigen plugin- infrastructuur: mogelijke oplossing

Voorgaande vereisten kunnen op verschillende wijzes gerealiseerd worden. De hier geformuleerde oplossing is attribute- gebaseerd:

- het onderscheid maken (differentiëren) tussen klasses die al dan niet als plugin moeten aanzien worden kan door middel van een attribuut. Dit attribuut ondersteunt twee properties: het al dan niet plugin zijn (isPlugin, bool), en de omschrijving van de eventuele plugin (description, String). Dit resulteert in een plugin- klasse met twee fields, twee (read only) properties en een constructor met twee argumenten;
  - indien een klasse dit attribuut niet heeft is de klasse geen plugin (default).
- indien een klasse het plugin attribuut heeft, en isPlugin is true, dan is dit een plugin klasse waarvoor we een extra menu item zullen voorzien (in de software die de plugin-assembly verwerkt). Indien op dit menu item wordt geklikt wordt een instantie van de plugin klasse aangemaakt en getoond
  - we veronderstellen (leggen op) dat het plugin attribuut enkel is toegepast geworden op Window- klasses;
  - het is *niet* de bedoeling om in het client- window een boekhouding tussen menuitems en plugin- klasses bij te houden: het menu-item zelf moet bij het klikken alle nodige bewerkingen onafhankelijk kunnen afhandelen.
    - dit laatste punt is belangrijk en lijkt soms in de totaal oefening verloren te gaan.

Jef Daels

5.4.3 Een eigen plugin- infrastructuur: implementatie

Zowel de plugin- assembly als het client- programma (het programma dat de plugins wil laden) moeten toegang hebben tot dezelfde plugin- attribuut klasse. Omdat er geen referentie tussen de plugin- en de client- assembly mag liggen (anders is de plugin assembly at compile time gekend en is de ganse plugin- opzet overbodig) en beide assemblies toch dezelfde attribuutklasse moeten kennen zijn we verplicht het plugin- attribuut in een apart project te plaatsen (waar zowel het plugin- project als het client- project naar refereren). We hebben dus minimum drie projecten! Omtrent die drie projecten kunnen we volgende vragen stellen:

- welk project type is elk project?
- welke referenties liggen tussen deze projecten?
- waarom hebben we eigenlijk drie projecten nodig? Indien het plugin- en client project beide dezelfde bron-file voor de attribuut klasse bevatten, is dit niet voldoende?
- formuleer een oplossing met maar twee projecten (perfect mogelijk trouwens, minder goed, naam gebaseerd).

Maak in dit derde project een attribuut klasse met volgende eigenschappen:

- volg de naamgeving guideline!
- voorzie de mogelijkheid om het plugin zijn en een plugin- omschrijving te bewaren, toe te kennen via de constructor en op te vragen via read only properties;
- maak het onmogelijk om dit attribuut toe te passen op andere code elementen dan klasse definities.

Maak in het tweede project (het plugin- project dat plugins aanlevert) een window aan en koppel wat logica aan het klikken op een knop. Plaats het plugin- attribuut op de MainWindow klasse en niet op de klasse clsPersoon.

Het clientproject dat de eventuele plugins zal laden gaat als volgt tewerk:

- bij het opstarten van een window dat de plugins ter beschikking stelt (de constructor van dit window is een goede plaats) lezen we alle assembly files gevonden op een vooraf afgesproken plaats (in mijn eigen oplossing is dit de plugins subdirectory van de toepassing)
  - System.IO is uw favoriete namespace voor dergelijke zaken;
  - gebruik een filter om enkel de assembly files op te vragen;
  - in een echte toepassing zou de plugin-directory configureerbaar moeten zijn.
- de types in elke *dll*- assembly worden overlopen. Indien ze voorzien zijn van een plugin attribuut (en isPlugin is *true*) voorzien we een extra menu item in het (main-)menu van dit window. Dit nieuwe menu-item is een instantie van een eigen menuitem klasse:

  - ze erft van de klasse *MenuItem*;
  - de omschrijving getoond in het menu is de description gevonden in het plugin attribuut;
  - ze implementeert een eventhandler zodat bij het klikken op een menu-item van deze klasse een istantie van het geassocieerde plugin-type wordt gemaakt en getoond;
  - bepaal zelf de nodige constructor argumenten.

5.4.4 Eigen plugin infrastructuur: demo

Gedemonstreerde elementen:

- het verwijderen van plugin assemblies verwijdert de plugins uit het menu;
- indien *isPlugin false* terug geeft wordt de eventuele plugin niet getoond;
- het verwijderen van het plugin attribuut zal (na hercompilatie) de plugin niet meer tonen in het menu;
- het plaatsen van het plugin- attribuut op een niet window klasse resulteert uiteraard in een foute werking. Een goed programma zal hierop niet crashen!
- er wordt in het window met de menu-items geen koppeling tussen de menu items en het plugin type voorzien: de volledige afhandeling om een plugin window te starten dient in de eigen plugin menu item-klasse geprogrammeerd te zijn.

5.5 Namespaces met interessante attributen

- System.Diagnostics: vooral debug en performance attributes;
  - ConditionalAttribute: zeer interessant wanneer u in uw code het onderscheid wensts te maken tussen Debug- en Release- uitvoeringen (klik nu op de link :)).
- System.Componentmodel: vooral component en control gerichte klasses (zie msdn);
  - onder andere tal van convertoren (niet echt IDE stuff);
  - licensing attributen (!);

5.6 Reflection: opgaves

- ontwerp een plugin infrastructuur (analoog met de demo);
- pas het fractalen programma aan zodat het tonen van de drietallige boom en Sierpinski als plugin in een programma wordt opgenomen.

Hoofdstuk 6

Runtime compilatie

6.1 Runtime compilatie: duiding

De meeste softwareontwikkelingen bestaan uit twee stappen: er wordt code geschreven (of gegenereerd) (stap één), waarna deze gecompileerd wordt tot een uitvoerbaar programma (stap twee). Eventueel zal een toepassing verschillende projecten bevatten die elk gecodeerd en gecompileerd worden. Dit beperkt uiteraard de mogelijkheden van de toepassing: wat niet op voorhand werd geprogrammeerd (eventueel in een PlugIn) kan niet uitgevoerd worden. Indien we een tekenprogramma wensen te ontwikkelen, waarbij de gebruiker het functievoorschrift intikt hebben we een probleem (we wensen niet zelf een wiskunde parser te schrijven). Ook indien we een toepassing maken waarbij de beveiliging van sommige onderdelen door de klant wordt bepaald kunnen we niet altijd alle mogelijkheden op voorhand voorzien. Sommige database toepassingen zullen een property list in een tabel bewaren (bv.: de talen waarin een produkt gekend moet zijn, zie ook de vertalingen demo): ook hier is het niet mogelijk om op voorhand alle properties in een klasse te compileren. In dergelijke situaties is het interessant om, gedurende de uitvoering van het programma, een extra stukje code te genereren, compileren en uitvoeren: *runtime compilatie* (ook wel on the fly compilatie geheten), zodat databinding zijn rol kan spelen.

Recentelijk werden de .NET compilers herschreven (Roslyn) zodat u indien u dat wenst de compiler informatie kan bekomen en wijzigen. Hoewel dit allerhande code manipulerende mogelijkheden biedt heb ik er nog geen gevonden waar ik het wenselijk acht om een eigen implementatie uit te werken.

6.1.1 Relevante namespaces

- *System.CodeDom.Compiler*

6.2 .NET compilaties starten

Zowel de $C^{\#}$als de VB.NET compilers maken deel uit van de CLR. Afhankelijk van de gegenereerde code ($C^{\#}$of VB.NET) zullen we dan ook de juiste compiler moeten oproepen. We zullen de code en het compilatieresultaat (de assembly) in memory houden zodat er geen files overblijven nadat ons programma eindigt.

6.2.1 Command line compilatie

Wellicht compileerde u tot nu toe altijd met behulp van Visual Studio. Het achterliggende compilatie proces (in welke volgorde compileren, welke compilatie parameters zijn nodig) wordt volledig verborgen (een voordeel). Om het compilatie proces duidelijker te maken zullen we in een eerste stap een volledig (console-) programma editeren, compileren en uitvoeren zonder gebruik te maken van Visual Studio.

6.2.1.1 Command line compilatie: $C^{\#}$

- maak een testfile abc.cs met een ascii- tekstverwerker (uw code files mogen enkel programma code bevatten, een tekstverwerker als Word is dus niet geschikt) en codeer volgende elementen ($C^{\#}$is case sensitive!):
  - voorzie een gepaste Main- methode (zie ook programmeren II: elk programma start door het uitvoeren van een static Main methode; of lees Main() and Command Line Arguments ($C^{\#}$Programming Guide)). Uiteraard moet u deze methode in een klasse coderen. Anders dan in JAVA moet de klasse naam niet identiek zijn aan de filenaam;
    - de klasse *String* zal enkel gekend zijn indien u ze prefixt met de namespace *System* of wanneer u een overeenkomstig *using* statement codeert;
  - schrijf het eerste runtime argument weg in de Console;

```
using System;
public class Test
        {
        public static void Main(String[] args)
                {
                Console.WriteLine(args[0]);
                }
        }
```

Figuur 6.1: Commandline compilatie: $C^{\#}$broncode

- start een Visual Studio command prompt (
  - win8: tik op het startscherm *tools* en kies de *native tools command prompt*;
  - win7: Start.All Programs.Microsoft Visual Studio xxx.Visual Studio Tools.Visual Studio Command prompt(xxx);
  - csc.exe is de $C^{\#}$compiler. U dient deze op te roepen met de juiste set parameters (*csc /?* toont u alle mogelijke opties met een summiere toelichting). Voor ons eenvoudige programma is de */out:* parameter eigenlijk overbodig (indien geen out- parameter wordt opgegeven heeft de resulterende assembly dezelfde naam als de bron-file);
  - met de target- parameter (niet in het voorbeeld) kan u aangeven of u een .exe (default) of .dll assembly wenst te bekomen;

Figuur 6.2: $C^{\#}$commandline compilatie en uitvoering

- indien ons programma gebruik zou maken van niet standaard bibliotheken moeten deze bij compilatie uiteraard ook worden meegegeven. Voor een gewoon WinForms of WPF programma wordt de parameterlijst zo groot (ongeveer elke project property komt terug als compiler optie), en is de compilatie volgorde zo belangrijk dat we dit zeker niet manueel wensen uit te voeren. Gedurende het programmeren zal Visual Studio dit voor ons doen. Indien u het proces wenst te automatiseren (om een groot project elke nacht volledig te hercompileren) kan u zich best wat verdiepen in MSBuild.

6.2.1.2 Command line compilatie: VB.NET

We volgen gelijkaardige stappen, maar dan voor VB.NET:

- VB.NET code:

```
imports System

public class test
   public shared sub main(args as String())
        Console.writeline(args(0))
   end sub
end class
```

Figuur 6.3: Commandline compilatie: VB.NET broncode

- VB.NET compilatie en uitvoering:

Figuur 6.4: VB.NET commandline compilatie voorbeeld

Het coderen en compileren is voor beide programmeertalen sterk gelijkaardig.

6.2.1.3 Een extraatje omtrent MSBuild

Het compileer proces van MSBuild gebeurt door middel van informatie gevonden in tekst files (zie ook MSBuild .Targets Files). Indien u hierin onderlegt bent kan u deze aanpassen (neen, u bent het wellicht niet). Aanpassen van deze files is de manier waarop PostSharp er in slaagt om het compilatie proces aan te passen zonder dat u als programmeur dient tussen te komen.

Jef Davis

6.2.2 Runtime compilatie: System.CodeDom.Compiler

Het is natuurlijk niet de bedoeling om voor normale sitaties een eigen compilatie proces uit te werken. Het is wel onze bedoeling om, vanuit een programma, programmacode te genereren en compileren (tot een .dll of .exe), en de bekomen assembly te verwerken zoals reeds werd gezien in reflection.

De .NET compilers zijn beschikbaar via de *System.CodeDom*- namespace:

System.CodeDom.Compiler Namespace

The **System.CodeDom.Compiler** namespace contains types for managing the generation and compilation of source code in supported programming languages. Code generators can each produce source code in a particular programming language based on the structure of Code Document Object Model (CodeDOM) source code models consisting of elements provided by the System.CodeDom namespace.

Classes

Class	Description
CodeCompiler	Provides an example implementation of the ICodeCompiler interface.
CodeDomProvider	Provides a base class for CodeDomProvider implementations. This class is abstract.
CodeGenerator	Provides an example implementation of the ICodeGenerator interface. This class is abstract.
CodeGeneratorOptions	Represents a set of options used by a code generator.
CodeParser	Provides an empty implementation of the ICodeParser interface.
CompilerError	Represents a compiler error or warning.
CompilerErrorCollection	Represents a collection of CompilerError objects.
CompilerInfo	Represents the configuration settings of a language provider. This class cannot be inherited.
CompilerParameters	Represents the parameters used to invoke a compiler.
CompilerResults	Represents the results of compilation that are returned from a compiler.
Executor	Provides command execution functions for invoking compilers. This class cannot be inherited.
GeneratedCodeAttribute	Identifies code generated by a tool. This class cannot be inherited.
IndentedTextWriter	Provides a text writer that can indent new lines by a tab string token.
TempFileCollection	Represents a collection of temporary files.

Figuur 6.5: *C#*System.CodeDom.Compiler namespace

Bemerk dat er geen aparte klasses per programmeertaal zijn:

- de klasse *CodeDomProvider* geeft toegang tot de .NET- compilers. De static methode *CreateProvider* heeft een argument waarmee wordt aangegeven voor welke programmeertaal we een compiler wensen te bekomen;
  - indien u twijfelt omtrent de taal-parametertekst (c# of csharp? VB of VB.NET? case sensitive?) kan u deze bekomen door de static methode *GetLanguageFromExtension* op te roepen (de file extentie moet u dan uiteraard wel kennen);
- *CompilerParameters* is handiger dan de commandline benadering: de verschillende parameters zijn niet als een lange tekst te formuleren maar zijn properties op een CompilerParameters object;
  - het toevoegen van een assembly kan gebeuren door middel van zijn naam: *System.Data.dll*
  - een assembly waarin een type XYZ zich bevind kan worden teruggevonden door *typeof(XYZ).Assembly.Location*

- *CompilerResults* bevat het compilatie resultaat. Dit bevat zowel de compilatie feedback (error, warning, ..) als de (eventueel) resulterende assembly;

6.3 CodeDom

De eerder aan bod gekomen oplossing verwacht broncode die gecompileerd kan worden: deze is natuurlijk programmeertaal afhankelijk. In een andere benadering kunnen we een programmeer boom maken waarin programmeer- entiteiten worden gehangen (types, methodes, variabelen, commando's, ..) waarvoor achteraf programma code (in de programmeertaal naar keuze) wordt gegenereerd:

```
CodeCompileUnit ccunit = new CodeCompileUnit();
CodeNamespace cns = new CodeNamespace("taalprog");
ccunit.Namespaces.Add(cns);

CodeTypeDeclaration klas = new CodeTypeDeclaration("klasnaam");
cns.Types.Add(klas);
klas.Attributes = MemberAttributes.Public;
```

Figuur 6.6: CodeDOM

Jef Daels

We spreken in deze context over een abstract syntax tree.

6.4 System.Reflection.emit

Voor de volledigheid vermelden we ook de System.Reflection.Emit Namespace. Deze laat toe om bij uitvoering *IL*- code genereren (normaal gezien het resultaat van een compilatie). Dit valt jammer genoeg buiten het bereik van deze cursus en mijn kennis.

6.5 Het Roslyn project

Een interessante ontwikkeling in het compilatie gebeuren van .NET is de ontwikkeling van het Roslyn project (momenteel nog in CTP (community technical preview)). Hierbij worden tussentijdse resultaten (de abstract syntax tree, koppeling tussen namen en objecten, ..) van het compilatie proces ontsloten naar ontwikkelaars toe. Dit biedt gigantisch veel mogelijkheden naar geautomatiseerde code processing (zeker naar rapportering toe, wellicht ook naar het automatisch aanpassen van code toe).

6.6 Assembly code inspecteren

Het resultaat van een .NET compilatie is (indien gelukt) een assembly. Deze bestaat uit *IL*(Intermediate language) code: de machinetaal van .NET. Deze assemblies kunnen door middel van disassemblers gemakkelijk gedecompileerd worden. Dit laat derden toe om uw broncode grotendeels te reproduceren wat meestal niet wenselijk is. Het gebruik van obfuscators kan de leesbaarheid van de gedecompileerde broncode sterk omlaag halen.

Deze problematiek is niet .NET specifiek: ook byte code (het java equivalent van IL- code) kan gedecompileerd worden. Ook voor C(++) output zijn er disassemblers te vinden. In de .NET context zal u merken dat u een assembly kan decompileren naar uw programmeertaal naar keus, wat ook de oorspronkelijke programmeertaal was.

6.6.1 ILDasm

ILDasm is de *IL* disassembler meegeleverd met *Visual Studio*. Deze kan gebruikt worden om *IL*- code te tonen in een textfile formaat. Op deze manier is het mogelijk om bijvoorbeeld de aanwezigheid van attributen na te kijken. Hoewel *ILDasm* zeker zijn plaats heeft (zie ook Ildasm.exe (MSIL Disassembler)) is het zeker niet de meest geschikte tool om de brontekst van een programma na te kijken:

Figuur 6.7: ILDasm output

6.6.2 ILSpy

ILSpy is het gratis antwoord waarmee werd gestart nadat de voorheen gratis Reflector software betalend werd. Voor zover ik beide oppervlakkig gebruikt heb vind ik ze sterk gelijkwaardig: beide laten toe om een door ons gecompileerd programma om te vormen naar een .NET-taal naar keuze.

Bij het downloaden van ILSpy kan u kiezen tussen de broncode en de gecompileerde bestanden. Ikzelf verkies meestal de broncode, maar het staat u uiteraard vrij om de binaries te installeren.

Gelieve na het installeren *ILSpy* eens op te starten en een programma te decompileren:

Figuur 6.8: ILSpy output

Bemerk dat in bovenstaande figuur VB.NET- code werd gegenereerd terwijl de oorspronkelijke programmatekst in $C^{\#}$werd geschreven.

6.6.3 Reflector

Deze (niet langer gratis) tool is (niet langer) wat u zoekt! Hiermee bent u in staat om een assembly te decompileren in *de programmeertaal naar keus*. Indien u de nodige plugins installeert kan u een assembly volledig decompileren tot een project (sommige namen zullen wel niet compileerbaar zijn, maar dat is voor kwaadwillige derden slechts een kleine beperking):

Figuur 6.9: Reflector output

Bemerk dat in bovenstaande figuur VB.NET- code werd gegenereerd terwijl de oorspronkelijke programmatekst in $C^{\#}$werd geschreven. In de programma code zijn de groene tekst elementen hyperlinks (niet beschikbaar in ILSpy): hiermee kan u navigeren naar de definitie van dit element (ongeveer zoals *F12* in Visual Studio).

6.6.4 Obfuscatie

Obfuscatie is een techniek om een assembly aan te maken die ook na decompilatie quasi onleesbaar is. De (gratis) versie die geïnstalleerd wordt met Visual Studio (onder tools) vertrekt van een compilatie resultaat en zal alle hierin gevonden namen (klasses, methodes, parameters, ..) *scramblen*, wat het geheel voor een complex programma quasi onleesbaar maakt. Het is uiteraard niet wenselijk om de broncode van het project zelf te scramblen, omdat u in dat geval geen aanpassingen meer zou kunnen doorvoeren.

Figuur 6.10: Obfuscated output via Reflector

Bemerk in bovenstaande figuur het hernoemd zijn van de controls. Veel routine namen zijn niet hernoemd geworden, wat spijtig is. Wellicht zal de betalende versie hier een betere obfuscatie mogelijk maken.

6.6.4.1 Confuser

Een betere obfuscater vindt u op Confuser. Het resultaat van deze obfuscater kan niet geopend worden in ILDASM en is totaal onleesbaar wanneer u bijvoorbeeld ILSpy gebruikt.

6.7 Opgaves

6.7.1 Een tekenprogramma

Het te realiseren programma moet aan volgende vereisten voldoen:

- de gebruiker kan zelf een functievoorschrift intikken (reden waarom we on-the-fly compilatie nodig hebben: we zullen dit functievoorschrift compileren zodat we het kunnen uitvoeren);
- het tekenen gebeurt in een usercontrol, waarbinnen de tekening geschaald wordt getoond: we rekken (of krimpen) de te tekenen figuur zodat de bovenkant van de figuur de bovenkant van de usercontrol raakt (idem voor de onderkant);

Figuur 6.11: Tekenen van een functie

6.7.2 ILSpy

Op het examen wordt verwacht dat u volgende manipulaties vlot kan uitvoeren:

- installeer ILSpy en pas deze toe op een assembly naar keus;
- demonstreer dat u kan decompileren naar verschillende programmeer talen.

6.7.3 Obfuscatie

Op het examen wordt verwacht dat u volgende manipulaties vlot kan uitvoeren:

- gebruik *Confuser* om een assembly naar keus te obfuscaten;
- gebruik de obfuscator (gevonden onder tools) om een assembly naar keus te obfuscaten;
- gebruik ILSpy om het obfuscated zijn van uw nieuwe assembly te demonstreren.

Jef Daels

Hoofdstuk 7

Backtracking

7.1 Backtracking: duiding

De eerste computers werden oorspronkelijk gebruikt om oplossingen te bieden voor problemen die mensen niet (snel genoeg) konden oplossen. Dit hoofdstuk behandelt het gebruik van een computer om problemen op te lossen waarvoor we zelf, zonder computer, geen antwoord kunnen formuleren. De gebruikte techniek heet *backtracking*: hij is niet op alle problemen toepasbaar, is soms hopeloos inefficient maar is een haalbare kaart om artificiële intelligentie te introduceren. In de laatste sectie zullen we backtracking gebruiken om AI in een spelletjes context te introduceren.

7.1.1 Relevante namespaces

7.1.2 Backtracking: principe

Backtracking is een benadering waarbij men, door het overlopen van alle mogelijke oplossingen, zoekt naar een beste (eerste, ..) oplossing. De implementatie van het algorimte is meestal recursief, met hierin volgende stappen:

- *stopconditie*: indien de oplossing werd gevonden geven we true terug;
- anders worden alle mogelijkheden voor deze stap in een lus overlopen:
  - voer één van de stap mogelijkheden uit;
  - indien de recursieve oproep voor de volgende stap slaagt hebben we een oplossing gevonden: we geven true terug;
  - indien de recursieve oproep faalt maken we de zopas gezette stap ongedaan en vervangen deze door één van de andere mogelijkheden, en herhalen de recursieve stap;
- indien geen enkele van de mogelijkheden tot een oplossing leidt moeten we concluderen dat de eerder gezette stappen het vinden van een oplossing blokkeren, en keren we terug naar de oproeper (return false), die dan op zijn beurt een volgende mogelijkheid kan onderzoeken.

7.2 N-Queens probleem

7.2.1 Probleemstelling, synchrone oplossing

Een gemakkelijk uit te leggen backtrack probleem is het N-Queens probleem: hoe plaats ik N koninginnen op een schaakbord zodat deze elkaar niet slaan. Twee koninginnen slaan elkaar indien ze:

- op dezelfde kolom staan;
- op dezelfde rij staan;
- uit de eerste koningin een diagonaal vertrekt waarop de tweede koningin zich bevindt.

Een geïllustreerde probleemstelling vind u op Eight queens puzzle. De toelichting van het algoritme gebeurt tijdens de les.

Figuur 7.1: NQueens oplossen

7.2.2 Asynchroon: BeginInvoke/4

Indien we grotere aantallen (bijvoorbeeld 30) koninginnen dienen te plaatsen neemt de berekening flink wat tijd in beslag. Om de interface hiermee niet te blokkeren zullen we de berekening van de oplossing op een andere thread laten uitvoeren. Hiertoe gaan we als volgt te werk:

- maak een delegate-class voor de zoekroutine;
- start een asynchrone oproep voor deze zoekroutine, en voorzie een callback waarmee het berekende resultaat in de UI kan getoond worden (u zal hoogst waarschijnlijk een state object nodig hebben);
- laat de asynchrone thread de berekende oplossing in de UI tonen (let op de UI-thread pitfall!).

Hint: om te vermijden dat gedurende de asynchrone zoekoperatie een volgende zoekoperatie gestart zou worden (er kan immers geklikt worden in de UI terwijl de zoekoperatie loopt) is het best om de UI-elementen die de zoekopdracht starten of specifiëren disabled worden geplaatst: het is immers niet wenselijk de zoek- informatie te wijzigen terwijl de vorige zoekopdracht nog loopt (de getoonde resultaten zouden niet langer overeenstemmen met de zichtbare criteria) net zomin als we een volgende zoekopdracht willen starten terwijl de vorige nog niet ten einde was. Als oefening kan eventueel een cancel- button voorzien worden.

7.2.3 Asynchroon: async/await

Hiervoor verwijs ik naar Await/Async verder in deze tekst. Deze is ook bruikbaar voor de Sudoku- toepassing die straks aan bod komt.

7.3 Sudoku's oplossen

Het oplossen van Sudoku's is niet altijd eenvoudig. Een computerprogramma kan u behulpzaam zijn, maar ontneemt u wellicht het liefhebbers plezier.

7.3.1 Sudoku: spelregels

Het invullen van een 9x9 sudoku gebeurt op volgende wijze:

- elk hokje van de matrix moet worden ingevuld met een geheel getal tussen 1 en 9 (grenzen inbegrepen);
- in een rij mogen geen dubbels voorkomen;
- in een kolom mogen geen dubbels voorkomen;
- in elk van de negen 3x3 hokjes (naast elkaar, niet overlappend) mogen er geen dubbels voorkomen;
- een opgave toont een 9x9 rooster met een aantal ingevulde velden:

		5	2	3	6		8	9
						2		
2								
3			4					
			8	9		3	2	7
	9		3			4	5	
		1	6	8				
					5			
		3	7					

Figuur 7.2: Sudoku spelregels: een opgave

- de oplossing is gevonden wanneer de speler het rooster correct aanvult:

1	4	5	2	3	6	7	8	9
6	3	7	9	1	8	2	4	5
2	8	9	5	7	4	1	3	6
3	1	2	4	5	7	6	9	8
4	5	6	8	9	1	3	2	7
7	9	8	3	6	2	4	5	1
5	2	1	6	8	3	9	7	4
9	7	4	1	2	5	8	6	3
8	6	3	7	4	9	5	1	2

Figuur 7.3: Sudoku spelregels: een oplossing

7.3.2 Software vereisten

- UI vereisten:
  - enkel numerieke input, beperkt tot één positie is mogelijk. Het is niet de bedoeling feedback te geven indien de gebuiker hiervan afwijkt, het is de bedoeling dat hij er niet kan van afwijken (zie ook TXTINT;
  - er is een intuitieve navigatie op basis van de pijltjes mogelijk;
  - er is kleurverschil tussen opgave en oplossing;
  - bij ingave van een foutief getal wordt dit (discreet) gekleurd.
  - de gebruiker kan de opgave niet wijzigen gedurende het oplossen
- Sudoku vereisten:
  - we kunnen opgaves ingeven en bewaren (op file), en deze later terug inlezen;
  - een opgave kan door het programma correct worden opgelost (indien er een oplossing bestaat).

Jef Daels

7.3.3 Sudoku oplos algoritme

De recursieve oproep *zoekOplossing(Cell cell,int[,]bord)* kent twee parameters:

- *cell* is de cell die we nu wensen in te vullen. Deze stuurt de stopconditie: indien de cel die we wensen in te vullen buiten het bord valt zijn alle bord cellen ingevuld en hebben we een oplossing;
  - *Cell* is een eigen klasse (we houden rij- en kolom- waardes bij (Integers). De klasse Point werd niet gebruikt omdat deze doubles bevat);
- *bord* bevat de bordsituatie zoals ze nu gekend is. Deze situatie hebben we nodig om het correct zijn van een nieuwe zet te beoordelen. Omdat in het bord de oplossing die we zoeken verder wordt opgebouwd bij elke recursieve stap spreken we van een *accumulerende parameter*.

- het return type is *boolean*: *true* indien een oplossing wordt gevonden, *false* anders.

Het algoritme gaat als volgt:

- indien *cell* buiten het bord valt zijn alle velden ingevuld en hebben we een oplossing in *bord*;
- overloop elke mogelijke waarde voor cell en voer volgende stappen uit:
  - vul de waarde in op de nieuwe positie;
    - indien dit een Sudoku probleem veroorzaakt gaan we over naar de volgende waarde;
    - indien dit geen probleem veroorzaakt voeren we een recursieve oproep uit voor de volgende cel. Indien deze oproep *true* teruggeeft vonden we een oplossing (in bord) en returnen we zelf ook *true*. In het andere geval (de recursieve oproep levert false) was het onmogelijk om de rest van het bord in te vullen op een correcte manier;
  - verwijder de zopas gezette waarde en probeer een volgende mogelijkheid;
- indien na het overlopen van alle mogelijkheden nog geen oplossing werd gevonden moeten we false terug geven: het was niet mogelijk de Sudoku verder op te lossen: we returnen false waardoor de oproeper zijn laatste poging ongedaan maakt en een andere poging zal ondernemen.

Heb speciaal aandacht voor volgende elementen:

- een zet die niet voldoet moet terug van het bord genomen worden;
- indien een oplossing wordt gevonden mag de bord parameter (die deze oplossing bevat) niet meer gewijzigd worden.

Dit algoritme wordt ook aan bord op basis van voorbeelden ontwikkeld. Ik heb niet de ambitie backtracking volledig schriftelijke toe te lichten.

7.3.3.1 Constraint satisfaction solver

Voorgaande is niet de enige manier om Sudoku's op te lossen. Een interessante alternatieve benadering wordt beschreven in Sudoku using Microsoft solver foundation. Het onderliggende algoritme zou opnieuw backtracking kunnen zijn.

7.4 Sudoku routed events introductie

Het is de bedoeling om pijltjes navigatie op het Sudoku bord toe te laten: we vangen het *PreviewKeyDownEvent* op en voeren de eventuele gepaste navigatie uit. Elke TextBox op het Sudoku-bord (er zijn er 81) kan de bron van dergelijk event zijn: in een klassieke benadering moeten minstens 81 eventhandlers gekoppeld worden. Indien men beslist andere controls dan TextBoxen te gebruiken moet men events van deze andere controls opvangen. Om in dergelijke situaties een elegante oplossing te bieden werden *routed events* geïntroduceerd:

- een WPF-control kan zich abonneren op een *routed event* gedefinieerd op een andere klasse (zoals een DependencyObject een DependencyProperty gedefinieerd op een andere klasse kan hebben):

```
public ccSudoku9x9()
    {
    this.Background = new SolidColorBrush(Colors.Transparent);
    initSudoku9x9();
    zetContextMenu();

    this.AddHandler(TextBox.TextChangedEvent, new TextChangedEventHandler( text_changed));
    this.AddHandler(TextBox.PreviewKeyDownEvent, new KeyEventHandler(keyDown));
    }
```

Figuur 7.4: Sudoku routed events demo

- de visual tree wordt eerst van container naar child elementen doorlopen (de *preview*-events): de *tunneling* fase;
- daarna wordt de visual tree van child naar container element doorlopen (de gewone event namen): de *bubbling* fase;
- indien een eventhandler (zowel tunneling als bubbling) het event afhandelt kan de verdere eventhandling afgebroken worden (handled=true op het argument e);
  - dit afbreken is op zijn beurt terug te omzeilen;
  - door het toepassen van templates en styles kan de visual tree behoorlijk complex worden. Toch wordt hij volledig doorlopen.
- hoewel *ccSudoku9x9* noch een *TextChangedEvent* noch een *PreviewKeyDownEvent* kent kan hij toch subscriben op deze routed events. In zoverre een element in de visual tree deze events afvuurt zal de ccSudoku9x9 control aan de bubbling- en tunneling fases deelnemen;
  - gebruik *e.OriginalSource* om te achterhalen welke control het event veroorzaakte.

7.5 Async methodes: TaskCompletionSource

Indien opfrissing omtrent await/async nodig verwijs ik naar de Random topics: awaitable methodes.
Zoals in het hoofdstuk omtrent delegates werd aangekaart kan met behulp van de klasse TaskCompletionSource een awaitable methode ontwikkeld worden.

Het .NET framework 4.5 bevat veel awaitable methodes, herkenbaar aan de suffix Async (zie ook Random topics: awaitable methodes). Om dergelijke methode asynchroon uit te voeren, en de rest van de routine definitie in een callback methode voor die oproep te plaatsen volstaat het om await voor de methode oproep te plaatsen. Meer nog, de rest code volgend op de awaited

oproep wordt automatisch terug op de huidige thread geplaatst (meestal de UI-thread).

Het gebruik van een awaitable methode is gemakkelijk, het opzetten van een awaitable methode is dit minder. De volgende tekst beschrijft hoe een awaitable methode kan aangemaakt worden.

7.5.1 Task- klasses

Een awaitable methode heeft *altijd* een Task- object terug. Indien het return type van een synchrone oproep *void* is, is het return type van de asynchrone versie *Task*. Indien het return type van de synchrone uitvoering T was geweest zullen we $Task< T >$ als resultaat type gebruiken van de asynchrone oproep.

Indien de oproep wordt *awaited* mag het resultaat van de oproep (te vinden in de *Result*-paramter van de Task) als resultaat van de await gebruikt worden (de compiler lost dit op):

```
Boolean gevonden = await ZoekOplossingAsync(oplossing,0,mogelijkeKolommen);
if (gevonden)
    ToonOplossing(oplossing);
```

Figuur 7.5: Awaiting een oproep

Zowel de NQueens- als Sudoku oplosser geeft als resultaat een Boolean terug. Het resultaat type van de Async- versie zal dan ook $Task< Boolean >$ zijn.

7.5.2 TaskCompletionSource- klasses

De *await*- infrastructuur dient geïnformeerd te worden omtrent het verloop van de asynchrone uitvoering: voltooiing, onderbreking, voortgang en dergelijke meer zijn belangrijk. Omdat deze informatie niet enkel via een callback kan gecommuniceerd worden werd een nieuwe generische klasse geïntroduceerd: *TaskCompletionSource*:

- de informatie omtrent de uitvoering van een asynchrone taak wordt verzameld in een TaskCompletionSource- object;
- dergelijk object wordt door ons programma als field gedefinieerd zodat het kan gemanipuleerd worden door onze routines;

```
private void ZoekOplossingCompleted(IAsyncResult ar)
    {
    Cel[] cellen = ar.AsyncState as Cel[];
    if (cellen[cellen.Length -1] != null) //er was e
        _tcsOplossing.SetResult(true);
    else
        _tcsOplossing.SetResult(false);
    }
```

Figuur 7.6: TaskCompletionSource: SetResult

- de start van onze asynhrone methode initialiseert dit field, start het asynchrone proces (met klassieke callback) en geeft de Task- property van dit object als resultaat terug;
- het is nu de bedoeling dat onze asynchrone methode (of zijn callback) na uitvoering dit object als completed of cancelled gaat instellen. De compiler heeft code gegenereerd die hierop inpikt om dan de logica die wachtte verder uit te voeren.

Wellicht kan een codevoorbeeld dit verduidelijken:

```
private delegate Boolean ZoekOplossingDelegate(Cel[] vorigeZetten, int
private TaskCompletionSource<Boolean> _tcsOplossing;
private Task<Boolean> ZoekOplossingAsync(Cel[] vorigeZetten, int huidi
    {
    _tcsOplossing = new TaskCompletionSource<Boolean>();

    ZoekOplossingDelegate del = new ZoekOplossingDelegate(ZoekOplossin
    del.BeginInvoke(vorigeZetten,huidigeRij,mogelijkeKolommen,ZoekOplo

    return _tcsOplossing.Task;
    }
```

Figuur 7.7: TaskCompletionSource demo

- ZoekOplossingDelegate is de delegate klasse om het zoeken asynchroon te programmeren;
- tcsOplossing is het resultaat van de awaitable methode. Ze wordt als field gedefinieerd zodat mijn logica deze kan aansturen;
- ZoekOplossingAsync is de awaitable methode (herkenbaar aan het return-type en de suffix van de methode). Deze methode zal intern een asynchrone oproep starten en geeft onmiddelijk een Task-object terug. De compiler genereert code die in de oproeper de rest van de logica uitvoert nadat deze task voltooid wordt. Het voltooien van de Task gebeurt door het resultaat in te stellen op het *TaskCompletionSource*- object waar de Task onderdeel van uitmaakt;

7.6 Het MiniMax algoritme (geen examen stof)

Backtracking wordt onder andere gebruikt als Artificiële Intelligentie algoritme bij het ontwikkelen van (turn-based) games. Het *MiniMax* algoritme is hiervan een voorbeeld. De Engelstalige wikipedia-entry voor MiniMax vind ik inhoudelijk sterker dan de Nederlandstalige.

7.6.1 MiniMax benadering voor vier op een rij

Het MiniMax algoritme realiseert volgende doelstelling: indien de speler aan zet niet wint met zijn volgende zet, dan minimaliseert hij de kansen van de andere speler. De andere speler zal bij het overlopen van zijn eigen mogelijkheden, indien hij niet direct wint, veronderstellen dat zijn tegenstander (speler 1) geen fouten maakt, en diens score maximaliseren, vandaar de naam van het algoritme. Omdat we bordsituaties met elkaar vergelijken (om een beste of slechtste situatie te kunnen kiezen) zullen we een BordWaarde- functie nodig hebben die de waarde van een bordsituatie weergeeft.

7.6.1.1 BordWaarde

- de parameters zijn die waardes nodig om een bordsituatie te beschrijven;
- het resultaat is een numerieke waarde, zodat vergelijkingen mogelijk worden:
  - +1: indien de speler aan zet uitspeelt;
  - -1: indien de tegenspeler uitspeelt;
  - 0: anders.
- de BordWaarde- functie zal intens gebruikt worden: elke mogelijke stap zal geëvalueerd worden door deze functie. Het is dan ook belangrijk om deze functie zo efficient mogelijk te maken.

Het is contra- intuitief om de bord- waarde te beperken tot 3 waardes: winst, verlies of onbeslist. Immers, wanneer we zelf vier op een rij spelen streven we bijvoorbeeld naar een situatie waar twee aansluitende rijen in een kolom tot winst leiden (zie kolom 3 in onderstaande figuur). In onze beoordeling van een bord is dit duidelijk een zeer goede situatie. MiniMax gewijs zal de AI, indien hij drie zetten vooruit *denkt* (rekent) ook merken dat in elke mogelijke invulling van een dergelijke situatie we een winnende zet kunnen realiseren, wat van dergelijke bord- situatie een winnende situatie maakt (+1). In onderstaande figuur werd de situatie in kolom 3 hierdoor als winnend herkent (bij de vorige zet) omdat de AI minstens drie zetten vooruit denkt:

Figuur 7.8: 4 op een rij: bordwaarde

7.6.1.2 BesteZet

De functie BesteZet berekent voor een gegeven bord situatie de best volgende zet voor een speler. In een vier op een rij context is het resultaat van de oproep een kolom nummer (int).

- het resultaat van de oproep is het kolomnr waarin de speler zijn fiche moet plaatsen. De parameters van de oproep worden bepaald door de nood van de hieronder geformuleerde logica;
- één van de argumenten van de oproep, *depth* geeft aan hoeveel stappen de computer vooruit mag denken. Hoe meer stappen de computer vooruit denkt, hoe langer de berekening zal duren.
- de speler aan zet (de maximizing speler) overloopt elke mogelijke kolom en voert volgende zaken uit:
  - plaats een fiche in deze kolom;
  - indien vier op een rij werd gevonden hebben we een winnende kolom: *return kolomnr*
  - indien geen directe winst wordt de waarde van deze kolom bepaalt als MiniMax(maximizingSpeler, minimizingSpeler, depth -1, ..).
  - indien de MiniMax- waarde kleiner is dan het huidige minimum wordt deze zet het nieuwe minimum (bewaar zowel de kolom als het minimax resultaat);
  - verwijder deze fiche terug
- de kolom met de kleinste MiniMax waarde (de slechtst mogelijke zet voor de tegenstander) wordt als resultaat teruggegeven.

Om wat random zetten aan het algoritme toe te voegen kunnen we volgende zaken aanpassen:

- we houden niet enkel de eerste beste zet bij, maar alle zetten die de huidige beste waarde als resultaat hebben;
- bij het teruggeven van de beste kolomwaarde wordt een random kolomnummer uit de lijst gekozen.

Jef Dacls

7.6.1.3 MiniMax methode

De Engelstalige wikipedia-entry voor MiniMax beschrijft het MiniMax algoritme als volgt:

```
function minimax(node, depth, maximizingPlayer)
    if depth = 0 or node is a terminal node
        return the heuristic value of node
    if maximizingPlayer
        bestValue := -∞
        for each child of node
            val := minimax(child, depth - 1, FALSE))
            bestValue := max(bestValue, val);
        return bestValue
    else
        bestValue := +∞
        for each child of node
            val := minimax(child, depth - 1, TRUE))
            bestValue := min(bestValue, val);
        return bestValue
```

Figuur 7.9: Minimax algoritme

- een *terminal node* is een zet die het spel afrondt: één van beide spelers realiseert vier op een rij. We veronderstellen dat enkel de speler die nu een fiche plaatst kan uitspelen, en dat zijn huidige zet deel uitmaakt van de oplossing (dat wil zeggen: we veronderstellen dat er zijn geen andere vier op een rij oplossingen zijn).
- u moet weten wie de maximizing speler is opdat u weet of de speler aan zet de mogelijkheden van zijn tegenstrander minimaliseert of maximaliseert;
- die MiniMax algoritme overloopt *alle* mogelijke zetten (tot eindspelsituaties) voor beide spelers, wat in veel spelsituaties een gigantisch aantal mogelijkheden betekent. Het beperken van die set mogelijkheden, zonder de beste oplossing te verliezen is dan ook zeer wenselijk.

7.6.2 Minimax optimalisaties

7.6.2.1 Alfa-Beta pruning

Indien u het MiniMax algoritme correct kon implementeren (geen onderdeel van de cursus), dan zal u wellicht geïnteresseerd zijn in Alfa-Beta pruning voor de MiniMax zoek boom. De uitleg en het bijhorende voorbeeld onderaan de webpagina kan ik niet verbeteren.

Een nadeel van maximaal pruning is het verliezen van de alternatieve beste zetten. Dit kan als gevolg hebben dat de AI zijn zetten nogal éénzijdig in het speelveld plaatst: indien de kolommen van links naar rechts worden doorgelopen zullen de eerste beste oplossingen zich altijd links ten opzichte van hun alternatieven bevinden. Omdat enkel de eerste beste oplossing wordt weerhouden wordt het speelveld vooral links ingevuld.

7.6.2.2 Multi- threading

Op een multi core toestel zal het single- threaded berekenen van de beste zet geen optimaal gebruik maken van de beschikbare rekenkracht. Het op een aparte thread plaatsen van de BesteZet berekening maakt deze berekening zelf niet multi threaded. Het multithreaded maken van vier op een rij lijkt voor de hand liggend, maar moet ik zelf nog eens uitwerken. Bij het multithreaded maken moet rekening worden gehouden met volgende elementen:

- de verschillende threads mogen elkaars situatie niet beïnvloeden. Elke variabele die gewijzigd wordt moet dan ook een copy van de oorspronkelijke waarde zijn.
  - veronderstel dat elke kolom-invulling in een aparte thread wordt gestart. Voor de eerste zet van de eerste speler worden dan onmiddellijk de acht eerste kolommen ingevuld. Indien ze op een gemeenschappelijk veld worden geplaatst heb ik onmiddellijk *acht* op een rij gerealiseerd. Bovenstaande redenering is van toepassing op *elke* variabele die door een thread zal worden aangepast;
  - bij het multithreaded maken van de oplossing worden verschillende deelzoekbomen parallel uitgevoerd. Dit conflicteert met de winst bekomen door de alfa- beta pruning waarbij een nieuwe deelboom pas wordt verwerkt nadat de beste oplossing uit de vorige deelboom werd berekend.
    - wellicht is het alfa-beta prunen van een multithreaded oplossing wel realiseerbaar, maar het is maar de vraag hoe groot de winst nog zal zijn. Indien de multithreaded oplossing pas even snel is als de pruned singled threaded oplossing hebben we gewoon resources onnodig verbruikt. Hier is nog ruimte voor wat onderzoek :).

7.7 Backtracking opgaves

- ontwikkel een N-queens oplosser;
  - Extended WPF Toolkit bevat tal van interessante WPF-controls, onder andere verschillende up/down controls. Wellicht kan u een aantal hiervan gebruiken bij het opzetten van uw oplossing;
  - voorzie een synchrone uitvoering, en demonstreer het blokkeren van de interface;
  - voorzie een asynchrone uitvoering (met cancel- mogelijkheid) en demonstreer de responsiveness van de interface (bijvoorbeeld door de cancel uitvoering);
  - voorzie een tweede asynchrone uitvoering die gebruikt maakt van het async/await mechanisme (zie cursustekst).
- ontwikkel een Sudoku oplosser.
  - voorzie een synchrone uitvoering. Het resultaat wordt zo snel bekomen dat een asynchrone uitvoering weinig zin heeft.

Jef Daels

Hoofdstuk 8

Dependency properties

8.1 Dependency properties: duiding

In een klassieke object georiënteerde omgeving definieert en implementeert een klasse de eigenschappen van haar instanties. In .NET wordt hiertoe een property- syntax gebruikt (we spreken hier van een CLR-property), in JAVA gebruikt men getter- en setter- routines. Zo zal de Text- property van de klasse TextBox in de TextBox-klasse zelf worden geïmplementeerd.

WPF introduceert de extra mogelijkheid om een property, gedefinieerd in klasse A, te koppelen aan instanties van klasse B. Dit vereenvoudigt sterk het opzetten van nieuwe infrastructuren: indien een Grid-klasse nood heeft aan het toekennen van *row* en *column* waardes aan subcontrols, dan kan de *Grid* klasse deze properties zelf introduceren. Deze dependency properties spelen ook een cruciale rol in WPF's databinding en animatie.

In praktijk kan u het dependency property systeem zien als een dictionary waarin voor objecten extra waardes worden bijgehouden. De infrastructuur laat toe om bij het instellen van de waardes custom code uit te voeren.

Attached dependency properties zijn XAML-georiënteerde dependency properties: om in XAML bruikbaar te zijn moet een dependency property een CLR wrapper hebben (die at run-time trouwens gebypassed wordt ..).

8.1.1 Relevante namespaces

- *System.Windows*
- *System.ComponentModel*
- *System.Windows.Media*
- *System.Windows.Data*

8.2 F1URL: een help infrastructuur

We zullen een F1-gebaseerd help systeem opzetten voor de UI-elementen in onze toepassing(en). Hiertoe wensen we de mogelijkheid te hebben om voor een willekeurige control een URL met de gewenste help bij te houden. Wanneer de gebruiker F1 klikt zal de webpagina van de control waarboven de cursor zich bevindt getoond worden. Indien de controls genest zijn zullen de eventuele verschillende pagina's getoond worden. Momenteel wordt in het .NET framework niet de mogelijkheid voorzien om declaratief een URL te koppelen aan een control (u kan natuurlijk zelf een dictionary systeem opzetten).

8.2.1 F1URL gebruik in XAML

```
<Window x:Class="URLDemo.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525"
        xmlns:src="clr-namespace:URLDemo"
        x:Name="window"
        src:F1URL.URL="abc"
        >
    <src:F1URL x:Name="grid">
        <Button Height="175" Name="btnOuter"
                HorizontalAlignment="Left" Margin="128,69,0,0"  VerticalAlignment="Top" Width="262"
                src:F1URL.URL="www.google.com" >
            <Button Content="Button" Height="67" HorizontalAlignment="Left"
                    x:Name="btnInner" VerticalAlignment="Top" Width="121"
                src:F1URL.URL="http://www.howest.be" />
        </Button>
    </src:F1URL>
</Window>
```

Figuur 8.1: Attached dependency property in XAML

In bovenstaande XAML-code zijn volgende elementen van belang:

- de namespace *src* hebben we zelf toegevoegd (gebruik de intellisense!);
- *src:F1URL* is een klasse die erft van *Grid*. De controls waarop de F1URL benadering moet werken zullen binnen dergelijke grid geplaatst worden (de *F1URL*- control zal het *KeyUp* event opvangen voor alle geneste subcontrols);
- de (attached) dependency property *F1URL.URL* wordt ingesteld voor het window en twee (geneste) buttons. De geneste buttons introduceren onmiddellijk een extra complexiteit: indien we F1 drukken als de inner button geselecteerd is, wensen we dan de help van de inner of van de outer button?

Wanneer we de Button- klasse erop nazien merken we dat er geen URL-property gekend is. Hoe werd het dan mogelijk om deze property toch in te stellen?

8.2.2 F1URL definitie

```
public class F1URL : System.Windows.Controls.Grid
    {
    public static readonly DependencyProperty URLProperty =
        DependencyProperty.RegisterAttached("URL", typeof(string), typeof(F1URL), new UIPropertyMetadata(null));

    //[System.Windows.AttachedPropertyBrowsableForType(typeof(UIElement))]
    [System.ComponentModel.Browsable(true)]
    public static string GetURL(UIElement target)
        {
        return (string)target.GetValue(F1URL.URLProperty);
        }

    public static void SetURL(UIElement target, string value)
        {
        target.SetValue(F1URL.URLProperty, value);
        }
```

Figuur 8.2: Attached dependency property definitie ($C^{\#}$)

Bemerk in bovenstaande code volgende zaken:

- een dependency property is een *static* field van het type *DependencyProperty*. De naam van het field eindigt met het woord *Property*;
- een nieuwe dependency property instantie wordt aangemaakt door middel van een *register(Attached)*- oproep met volgende argumenten:
  - de naam in de registratie is identiek aan de fieldnaam, met weglating van de suffix *Property*;
  - het property type definieert het type van de property (een tekst- property heeft hiervoor als parameter *typeof(string)*);
  - het ownertype definieert de klasse die als prefix voor de property zal gebruikt worden: *F1URL*.URL (typeof(F1URL) is hierin het owner type);
  - extra informatie omtrent het verwerken van de property waardes.
- we voorzien de dependency property van (static) Get- en Set- wrapper routines (de hoofdletters *G* en *S* zijn belangrijk!). De Get- en Set- namen zijn (op de drie-letter prefix na) identiek aan de naam gebruikt in de register methode. Zonder deze wrappers is de dependency property niet gekend in XAML.
  - merkwaardig genoeg zal XAML de wrappers volledig bypassen, zoals beschreven in XAML Loading and Dependency Properties;
- indien we de Get- wrapper voorzien van een *Browsable*- attribuut zal deze dependency property ook getoond worden in de property list in Blend (in Visual Studio 2010 wordt de property getoond indien hij in de XAML-code aanwezig is);
- de snippet *propdp* is standaard in Visual Studio aanwezig om de dependency property definiërende code te genereren(*propa* voor attached dependency properties).

8.2.3 F1URL infrastructuur

We definieerden de dependency property *F1URL.URL* om een help infrastructuur op te zetten. Indien een gebruiker F1 duwt wensen we een help-pagina te tonen. Om het key-up event van een willekeurige control op te vangen gaan we als volgt te werk:

- we maken een eigen Grid-klasse versie waarbinnen alle controls die F1URL-ondersteuning nodig hebben geplaatst zullen worden;
  - de hier voorgestelde oplossing werkt enkel voor controls binnen onze eigen Grid-klasse. Deze beperking is overbodig wanneer we echte behaviors zullen ontwikkelen;
  - de huidige opzet is geschikt om bubbling (routed) events toe te lichten;
- de container control zal het bubbling *keyup*-event opvangen en actie ondernemen om de nodige help-topics te tonen:

```
public class F1URL : System.Windows.Controls.Grid
    {
    "dep prop URL"
    public F1URL() : base()
        {
        this.KeyUp += new System.Windows.Input.KeyEventHandler(URLF1_KeyUp);
        }

    void URLF1_KeyUp(object sender, System.Windows.Input.KeyEventArgs e)
        {
        if (e.Key == System.Windows.Input.Key.F1)
            verwerkF1(e.OriginalSource as FrameworkElement);
        }

    private void verwerkF1(FrameworkElement fwe)
        {
        if (fwe == null) return;
        string naam = fwe.Name;
        string url = (string) fwe.GetValue(F1URL.URLProperty);
        if (url != null)
            toonURL(url,naam);
        verwerkF1(fwe.Parent as FrameworkElement);
        }

    private void toonURL(string URL,string naam)
        {
        try
            {
            System.Diagnostics.Process.Start(URL);
            }
        catch (Exception ex)
            {
            }
        }
    }
```

Figuur 8.3: F1 infrastructuur ($C^{\#}$)

- de naam van het frameworkelement (een control parent klasse) is aanwezig omwille van debug-redenen;
- de *Process*-klasse start automatisch uw favoriete browser (bemerk dat *Process* ook het geschikte programma zal starten indien het argument geen URL zou zijn);

- de infrastructuur verwerkt alle geneste controls.

Uiteraard is deze eenvoudige benadering niet helemaal naar wens: zo willen we help kunnen opvragen omtrent een knop zonder(!) dat we deze dienen aan te klikken: het aanklikken start namelijk het proces waarvoor u de help wenst op te vragen, dit is zeker niet wenselijk. De infrastructuur moet rekening houden met de positie van de cursor en niet met de geselecteerde control. De lesdemo illustreert dit en maakt hiertoe gebruik van hit- testing om de control onder de cursor op te zoeken.

8.3 Dependency properties: technisch

- een dependency property is een static DependencyProperty- field in een klasse;
- een dependency property heeft een *owner type*. Dit is niet noodzakelijk de klasse waarbinnen de property wordt gedefinieerd. Een dependency property kan trouwens meerdere owner types hebben (we maken dan gebruik van de addOwner methode van de DependencyProperty klasse).
- een attached dependency property is een dependency property bruikbaar via XAML. Opdat XAML de property zou kennen moeten de naamgevingsregels gevolgd worden!
- bij het registreren van een dependencyproperty kan optioneel extra informatie worden meegegeven:
  - een default waarde;
  - verschillende routines (allen in de vorm van delegates):
    - propertyChangedCallback: wordt uitgevoerd wanneer de property waarde wijzigt (niet noodzakelijk bij elke toekenning!);
    - coerceValueCallback: wordt gebruikt om een eventuele nieuwe waarde om te vormen tot een andere waarde;
    - validateValueCallback: valideert de geldigheid van de nieuwe waarde voor deze klasse (niet in combinatie met de andere state elementen van het target object waaraan de value wordt toegekend). Het boolse resultaat wordt onmiddellijk gebruikt in de XAML-editor om de eventuele incorrecte waarde aan te geven. Deze routine wordt uitgevoerd *na* de coerce- routine;
- het definiëren van een dependency property is niet voldoende om deze property voor een object te kunnen instellen: dependencyproperties zijn enkel instelbaar op instanties van klasses die erven van DependencyObject!

Het is een aanrader om als delegate oefening eens een uitgewerkte URL- attached dependency property aan te maken met volgende elementen:

- een URL is slechts geldig indien hij start met *http://* (hoofdletterongevoelig);
- voor elke TextBox control zal de url *http://www.howest.be* worden omgevormd tot *http://www.nmct.be*.

Stel in XAML-code een aantal URL's in om uw implementatie te testen. Voorzie in uw testen volgende situaties:

- voorzie een URL-waarde die niet start met de gevraagde prefix, en observeer het resultaat in XAML en bij het opstarten;
- voorzie een textbox van een om te vormen URL;
- pas de coerce routine aan zodat de nieuwe waarde niet voldoet aan de validate routine en observeer het resultaat.

8.4 Dependency property XAML binding

Via XAML-code kunnen we enkel dependency properties *binden*. Indien we proberen een binding op te zetten voor een gewone (CLR) property, dan krijgen we bij uitvoering volgende foutmelding:

```
<src:ccNQueens Grid.Row="1" x:Name="grdOplossing"
               DimensieCLR="{Binding Value, ElementName=iudQueens}"/>
```

XamlParseException occurred

A 'Binding' cannot be set on the 'DimensieCLR' property of type 'ccNQueens'. A 'Binding' can only be set on a DependencyProperty of a DependencyObject.

Figuur 8.4: CLR binding error

8.5 Dependency property syntax

Het instellen van een dependency property kan in $C^{\#}$op twee manieren gecodeerd worden:

- *target.setValue(ownertype.depprop, value)*: het object gebruikt de geërfde methode setValue om een dependencyproperty waarde te bewaren;
- *ownertype.Setdepprop(target,value)*: de static setter routine wordt gebruikt;

```
anWidth.SetValue(Storyboard.TargetNameProperty,btnInner.Name);
anWidth.SetValue(Storyboard.TargetPropertyProperty, new PropertyPath( Button.WidthProperty));
```

Figuur 8.5: Dependency property syntax 1 ($C^{\#}$)

Jeff Dals

```
Storyboard.SetTargetName(anWidth, btnInner.Name);
Storyboard.SetTargetProperty(anWidth, new PropertyPath(Button.WidthProperty));
```

Figuur 8.6: Dependency property syntax 2 ($C^{\#}$)

Er bestaan uiteraard gelijkaardige notaties voor het opvragen van de dependency property. Beide notaties komen aan bod gedurende het bespreken van de animaties.

8.6 Dependency properties en animaties

WPF ondersteunt een animatie infrastructuur (niet de topic hier) gebaseerd op dependency properties: het wijzigen van een dependency property met impact op de visualisatie zal deze visualisatie onmiddellijk wijzigen. Daarbovenop komen een aantal (animatie-) klasses die gebruikt kunnen worden om waardes van start- waarde naar een eind- waarde te laten evolueren gedurende een tijdsinterval (met easing). In combinatie met Blend maakt dit het vlot opzetten van tijdlijn animaties en storyboards mogelijk.

In dit cursusdeel zullen we de animaties implementeren door middel van $C^{\#}$- code. Door Blend gegenereerde code is XAML code. Het is *niet* de bedoeling om een impliciete voorkeur te laten blijken.

8.6.1 DoubleAnimation

De breedte van een control is een double waarde. De DoubleAnimation klasse voorziet de mogelijkheid om een double waarde geleidelijk te evolueren tusen een start- en eindwaarde. Indien men dan de breedte- eigenschap van een control koppelt aan deze double- evolutie bekomt men een animatie:

```
private void animeerBreedte()
    {
    DoubleAnimation anWidth = new DoubleAnimation(btnInner.Width, btnInner.Width * 1.1,
        new Duration(new TimeSpan(0, 0, 2)));
    anWidth.SetValue(Storyboard.TargetNameProperty,btnInner.Name);
    anWidth.SetValue(Storyboard.TargetPropertyProperty, new PropertyPath( Button.WidthProperty));
    anWidth.AutoReverse = true;
    anWidth.RepeatBehavior = RepeatBehavior.Forever;

    Storyboard sb = new Storyboard();
    sb.Children.Add(anWidth);
    sb.Begin(this);
    }
```

Figuur 8.7: Breedte animatie ($C^{\#}$)

Op basis van bovenstaande code formuleren we volgende opmerkingen:

- de DoubleAnimation instantie zelf weet niet op welke eigenschap van welk object ze zal worden toegepast;
- het koppelen van een DoubleAnimation aan een eigenschap van een control gebeurt door middel van twee dependency properties, gedefinieerd in de StoryBoard klasse:
  - *StoryBoard.TargetNameProperty* wordt gebruikt om de naam te bepalen van het element (de control) waarvan een eigenschap geanimieerd zal worden;
  - *StoryBoard.TargetPropertyProperty* wordt gebruikt om de dependencyproperty die geanimeerd zal worden te identificeren op basis van zijn PropertyPath.
- animaties worden (hier) als onderdeel van een StoryBoard uitgevoerd;
- het starten van een StoryBoard heeft nood aan een visueel element waarbinnen de te animeren controls worden gezocht.
  - in *WinRT*-toepassingen is het niet nodig de *Begin*- methode van een *StoryBoard* van parameters te voorzien.

8.6.2 ColorAnimation

Het animeren van de achtergrondkleur van een control gebeurt op gelijkaardige wijze maar is verschillend op volgende punten:

- de achtergrond van een control is een Brush (en geen Color!). Het animeren van de achtergrondkleur gebeurt dan ook door de kleur van een (SolidColor)Brush te animeren (we hebben een ColorAnimation- class, en geen SolidColorBrushAnimation class);
- het PropertyPath, nodig om een achtergrondkleur te animeren bestaat uit twee stukken: het eerste stuk kiest de Background property, het tweede stuk kiest daarvan de SolidColorBrush.Color property. De code zal enkel werken indien de background een SolidColorBrush waarde bevat. Indien dit niet het geval is (de achtergrond is niet ingesteld, of is een ander soort Brush), dan zal de achtergrond uiteraard niet geanimeerd worden (de gezochte eigenschap is niet aanwezig). Dit genereert trouwens *geen* uitvoeringsfout.

```
private void animeerBackground()
    {
    ColorAnimation anKleur = new ColorAnimation(Colors.Red,new Duration(new TimeSpan(0,0,2)));
    Storyboard.SetTargetName(anKleur, btnInner.Name);
    Storyboard.SetTargetProperty(anKleur,new PropertyPath("(Panel.Background).(SolidColorBrush.Color)"));
    anKleur.AutoReverse = true;
    anKleur.RepeatBehavior = RepeatBehavior.Forever;
    Storyboard sb = new Storyboard();
    sb.Children.Add(anKleur);
    sb.Begin(this);
    }
```

Figuur 8.8: Background animatie

Jef Daels

8.6.3 Transformatie animaties

Het tranformeren van een figuur bestaat veelal uit drie onderdelen: een schaling, een rotatie en een skewing. Voor elk van deze onderdelen bestaat er een transformatie klasse. Verschillende transformaties kunnen worden gebundeld tot een transformatiegroep:

```
private void transformeer()
    {
    ScaleTransform scale = new ScaleTransform(0.75, 1.1);
    RotateTransform rot = new RotateTransform(90);
    SkewTransform skew = new SkewTransform(30, 30);
    TransformGroup tg = new TransformGroup();
    tg.Children.Add(scale);
    tg.Children.Add(rot);
    tg.Children.Add(skew);

    btnOuter.RenderTransform = tg;
    }
```

Figuur 8.9: Transformatie van een Button ($C^{\#}$)

- in veel gevallen is het gemakkelijker om een CompositeTransform te gebruiken in plaats van een transformatiegroep.
- bovenstaande code is nog geen animatie: er wordt een eenmalige transformatie toegepast op een control die daarna onveranderd blijft;

Een animatie zou er in bestaan om bijvoorbeeld de schaling en/of de draaihoek te animeren. Dit zijn properties van de zopas gedefinieerde objecten rot en scale. Deze objecten zijn niet bij naam gekend door de animatie infrastructuur (in de vorige animaties kon het te animeren object (een control) via zijn naam geïdentificeerd worden). Om properties van deze nieuwe objecten toch te kunnen animeren zullen we een naam voor deze objecten registreren. Na registratie kunnen ze via hun naam in een TargetNameProperty context gebruikt worden:

- in een *WinRT* toepassing kunnen we een object animeren zonder zijn naam te kennen door middel van de *TargetProperty*- dependency property van de StoryBoard klasse.

```
private void transformeer()
    {
    ScaleTransform scale = new ScaleTransform(0.75, 1.1);
    RotateTransform rot = new RotateTransform(90);
    SkewTransform skew = new SkewTransform(30, 30);
    TransformGroup tg = new TransformGroup();
    tg.Children.Add(scale);
    tg.Children.Add(rot);
    tg.Children.Add(skew);

    btnOuter.RenderTransform = tg;

    this.RegisterName("myrot", rot);
    this.RegisterName("myscale", scale);

    DoubleAnimation anAngle = new DoubleAnimation(rot.Angle, rot.Angle + 360,new Duration(new TimeSpan(0,0,4)));
    anAngle.SetValue(Storyboard.TargetNameProperty, "myrot");
    anAngle.SetValue(Storyboard.TargetPropertyProperty, new PropertyPath(RotateTransform.AngleProperty));
    anAngle.RepeatBehavior = RepeatBehavior.Forever;

    Storyboard sb = new Storyboard();
    sb.Children.Add(anAngle);
    sb.Begin(this);
    }
```

Figuur 8.10: Transformatie animatie ($C^{\#}$)

Jef Daels

8.6.4 Animaties en XAML

In veel gevallen is het gemakkelijker om animaties aan te maken met behulp van een design tool (*Blend*) dan via programmeer code. De door Blend gegenereerde code is XAML-code die buiten het bestek van dit hoofdstuk valt.

8.6.5 $C^{\#}$of XAML animaties?

Het animeren via code wordt naar mijn aanvoelen minder aangemoedigd dan het animeren via XAML (met behulp van tools zoals Blend). Toch meen ik dat een aantal situaties te herkennen zin waarbij $C^{\#}$- animatie een elegante oplossing kan bieden:

- animatie van dynamisch toegevoegde controls;
- animaties waarvoor de toe te passen waardes afhankelijk zijn van de run-time situatie en niet at-design time gekend zijn. Voorbeelden hiervan zijn:
  - het zweven van een fiche naar een speelveld op een spelbord;
  - het proportioneel verschuiven van elementen binnen een control, die niet altijd even groot moet zijn;

– een aantal basis- animaties (zoals bv. opacity) heb ik als extention method gedefinieerd op de control klasse. Het lijkt me dan ook weinig zinvol om deze animatie in XAML te definiëren.

Bovenstaande situaties zijn implementeerbaar via XAMl, maar behoeven dan meestal andere extra $C^{\#}$-code elementen zoals TypeConvertor of extra properties.

8.7 DependencyProperties in XAML: TypeConverters

Indien we in XAML een dependency property instellen moeten we ons beperken tot tekst. Dit is bruikbaar voor tekstwaardes, maar niet indien het resultaat type een niet-tekst type is. Om dit toch mogelijk te maken worden typeconvertoren gebruikt. Deze zijn in staat om op basis van een string een instantie van het property type te construeren (een vorm van deserialisatie). WPF bevat reeds een aantal typeconvertoren voor frequent voorkomende conversies (getallen, kleuren, ..) maar in sommige gevallen is het toch nodig om ook zelf een typeconverter te implementeren.

8.7.1 IValueConverter interface

Deze interface bevat een *Convert* en *ConvertBack* methode (in onderstaand voorbeeld is enkel de Convert van belang).

Het *ValueConversion* attribuut wordt gebruikt om het systeem te informeren omtrent de geïmplementeerde typeconversies. Onderstaande code codeert een converter die tekst omvormt tot een *LineairGradientBrush*. Er wordt verondersteld dat de twee kleurnamen in het Engels, en hoofdlettergevoelig geschreven worden:

```
[System.Windows.Data.ValueConversion(typeof(string),typeof(System.Windows.Media.LinearGradientBrush))]
public class LinearGradientConverter : System.Windows.Data.IValueConverter
    {
    public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
        {
        string tekst = value as string;
        if (tekst == null) return null;

        string[] kleuren = tekst.Split(new char[]{','});
        BrushConverter bc = new BrushConverter();
        Color kleur1 = ((SolidColorBrush) bc.ConvertFrom(kleuren[0])).Color;
        Color kleur2 = ((SolidColorBrush)bc.ConvertFrom(kleuren[1])).Color;
        return new LinearGradientBrush(kleur1, kleur2, 45);
        }

    public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
        {
        throw new NotImplementedException();
        }
    }
```

Figuur 8.11: ValueConverter definitie ($C^{\#}$)

In XAML kunnen we dan volgende code noteren:

```
<Window.Resources>
    <src:LinearGradientConverter x:Key="lgc"></src:LinearGradientConverter>
    <s:String x:Key="redgreen">Red,Green</s:String>
</Window.Resources>
<src:F1URL x:Name="grid">
    <Button Height="129" Name="btnOuter"
            HorizontalAlignment="Left" Margin="128,69,0,0"  VerticalAlignment="Top" Width="262"
            src:F1URL.URL="http://www.google.com"
            Background="{Binding Source={StaticResource ResourceKey=redgreen}, Converter={StaticResource ResourceKey=lgc}}" >
    </Button>
```

Figuur 8.12: ValueConverter XAML ($C^{\#}$)

Hier wordt de achtergrond van een Button (een Brush dus) ingesteld door middel van een tekst (Red,Green) en een typeconverter. Het resultaat is een LineairGradientBrush.

Jammer in de XAML notatie is dat de tekst-waarde momenteel als static resource gedefinieerd wordt (misschien kan dit anders?)

8.7.2 Default dataconverters

Veelal lijkt de convertie van tekst in XAML naar een overeenkomstig object spontaan te gebeuren, zonder expliciete vermelding van de nodige type converter. Om dit te realiseren wordt een klasse voorzien van een default typeconverter (door middel van het TypeConverterAttribute attribuut). De typeconverter klasse wordt jammer genoeg anders opgebouwd dan de *ValueConverters* die eerder aan bod kwamen. Om een klasse als impliciete typeconverter te kunnen gebruiken gaan we als volgt te werk:

- de klasse die de conversie zal uitvoeren *moet* erven van de klasse *TypeConverter* (de interface *IValueConverter* is hier onbelangrijk);
- het is belangrijk om minstens twee methodes te overriden:
  - *CanConvertFrom* die default *false* terug geeft die aangeeft of het gevraagde brontype wel geschikt is om te converteren (in XAML is het brontype *String*);
  - *ConvertFrom* is de methode die de effectieve conversie uitvoert.

Wanneer een klasse een default typeconverter kent die tekst kan omvormen tot een instantie van de klasse, dan is het niet langer nodig om in de XAML-code een expliciete converter te voorzien.

8.7.2.1 TryParse

Wanneer ik een conversie van tekst naar een type X moet coderen zal ik het type X voorzien van een *TryParse* methode:

- het resultaat type is Boolean: true indien de conversie lukt, false anders;

Jef Daels

- de eerste parameter bevat de tekstwaarde die geconverteerd moet worden;
- de tweede parameter (een referentie parameter) bevat het resultaat na de conversie (ongewijzigd indien de conversie niet lukt).

Het gebruik van *TryParse* is in .NET reeds op vele plaatsen terug te vinden (Integer, Double, DateTime, IPAdress, ..). Het implementeren van dergelijke methode is vooral een conventie, en zeker geen technische vereiste.

8.8 Dependency properties en OnRender

Indien een dependencyproperty belangrijk is voor het visuele aspect van de dependency objecten waarop de property wordt toegepast dan kan dit in de meta informatie worden opgenomen (bij het registreren van de dependency property). Indien dit het geval is zal het wijzigen van de dependencywaarde de *OnRender*- methode van het object waarvoor de dependecyproperty wijzigt triggeren. Dit op zijn beurt heeft als gevolg dat u het hertekenen niet moet oproepen via de propertychanged callback methode. Indien u dit wenst uit te pluizen (helemaal niet moeilijk) kan u zoeken op *FrameworkPropertyMetadata.AffectsRender* en *FrameworkPropertyMetadata.AffectsMeasure*.

8.9 Veel voorkomende fouten

- niet volgen van de naamgevings conventies (ook hoofdletters!);
- *null* als default waarde van een value type;

8.10 Extra lectuur

- Wie een interessant voorbeeld met vele depencendy properties wenst na te lezen verwijs ik graag door naar Render Text On A Path With WPF
- het aanmaken van een eigen dependency property wordt beschreven in Custom Dependency Properties
- een goede technische beschrijving omtrent attached dependency properties

8.11 Dependency property: opgaves

- animeer via code een control:
  - zijn afmetingen (breedte en grootte);
  - zijn achtergrondkleur;
  - roteer de control;
- maak een extention method waarmee u de opacity van een control (elke control) kan animeren: FadeInOut(double opacity);
- implementeer de *F1URL* infrastructuur en demonstreer dat deze werkt.
- voorzie de NQueens toepassing van een *Dimensie*- dependency properties, en bind deze aan de IntegerUpDown control zodat de code-behind leeg wordt;

```
<src:ccNQueens Grid.Row="1" x:Name="grdOplossing"
               src:ccNQueens.Dimensie="{Binding Value, ElementName=iudQueens}"  />
```

Figuur 8.13: XAML binding dependency property

Jef Daels

Hoofdstuk 9

Templated controls

Eventuele oefening: een ccDatePicker

9.1 Templated controls: duiding

Een betrachting in moderne WPF toepassingen is het scheiden van logica en visualisatie (de wenselijkheid hiervan hangt af van de situatie). Dit wordt in veel gevallen gerealiseerd door het gebruik van XAML (usercontrols, templates, styles, ..) die gebonden wordt aan (dependency) properties gevonden in de logica (zie ook . Dit hoofdstuk behandelt extra mogelijkheden (bovenop het binden van properties) om interactie tussen beide lagen mogelijk te maken.

- de term *templated controls* duidt hier op die controls die in de visualisatie van het dataelement zijn opgenomen.
- dit hoofdstuk is geen pleidooi voor of tegen de scheiding tussen logica en view. Het al dan niet wenselijk zijn van die scheiding is projectafhankelijk, en de factoren zijn niet enkel puur technisch: vraag van de klant, huisstijl, beschikbare tijd, .. zijn evenzeer elementen die van belang zijn.

In dit hoofdstuk krijgt u zeer weinig code te zien. Het is dan ook de bedoeling om, met deze tekst als leidraad, zelf de code op te stellen: het is een goede oefening op delegates en dependency properties.

9.2 Relevante namespaces

9.3 CaptionControl

Wanneer we een WPF-toepassing maken is het niet zo eenvoudig om de titelbalk bovenaan een window aan te passen (u moet de windows themes wijzigen, wat zijn effect heeft op elke toepassing). In dit hoofdstuk zullen we een eigen control maken die als titelbalk kan dienen, maar waarvan we de eigenschappen (hoogte, lettertype, gebruikte images in de controlbox, ..) volledig zelf kunnen aansturen. Het is de bedoeling een custom control te ontwikkelen, zodat de visualisatie volledig wijzigbaar is. Het voorbeeld illustreert zowel het gebruik van PARTS als STATE's.

9.3.1 PARTS

9.3.1.1 Caption control via PARTS: inleiding

Als basis voor deze tekst gebruiken we MSDN: Creating a Control That Has a Customizable Appearance.

Wanneer u een custom control maakt geeft u de visualisatie van uw control uit handen: de gebruikende programmeur (designer) is in staat om de volledige visualisatie aan te passen. Indien u rekent op de aanwezigheid van een visueel element om een correcte verwerking te kunnen garanderen (onze CaptionControl verwacht dat er 'iets' is waarmee het window kan gesloten worden) is het wenselijk om dit aan de gebruiker (en zijn ontwikkel software: Blend of Visual Studio) duidelijk te maken. Dit gebeurt op twee manieren:

- PART_ prefix (ik vind dit niet terug als guideline, en wel een Pro WPF in VB 2010, pagina 567 die stelt dat deze naamgeving sinds de introductie van VSM verouderd is);
- *TemplatePart* attributen op de customcontrol.

In de *OnApplyTemplate* methode van de custom control kunnen we controls, aangemaakt door toepassing van de template opvragen en via events koppelen aan gewenst gedrag. U moet uw code voorzien op de eventualiteit dat de gebruiker de vereiste parts niet aanbood met de gewenste naam. In dat geval is het mogelijk dat een stuk functionaliteit niet langer aanwezig is, maar uw toepassing mag hierdoor natuurlijk geen fout veroorzaken!

Figuur 9.1: Window met eigen caption

9.3.1.2 Caption Control via PARTS: uitwerking

- maak een nieuw WPF project, en plaats de WindowStyle property van het aanwezige Window op *None*. Hierdoor zal het Window bij het opstarten geen caption meer vertonen. Het is de bedoeling om nu zelf een caption control te maken die dezelfde functionaliteit aanbiedt en beter aanpasbaar is;
  - hint: u kan dit venster nog steeds afsluiten met de *Alt-F4* toetscombinatie;

Figuur 9.2: Window met WindowStyle none

- voeg via Visual Studio een *Custom Control ccCaption* toe aan het project. Hierdoor wordt een gepaste static constructor en een bijhorend theme aangemaakt. Save en compileer het project waarna u het in Blend opent;
- editeer de resource gevonden in *Generic.xaml*:
  - voorzie een grid met 4 kolommen;
  - de drie laatste kolommen worden gebruikt om de *minimize*, *maximize* en *close* functionaliteit aan te bieden. Omdat de te gebruiken images vierkant zijn zullen deze drie laatste kolommen net zo breed maken als ze hoog zijn (we binden de *Width* property aan de *ActualHeight* property van hun *Grid Ancestor*). De eerste kolom neemt de rest van de beschikbare plaats in beslag en zal de te tonen titel bevatten.
- plaats een Image in elk van de drie laatste kolommen, met als naam PART_Minimize, PART_Maximize, PART_Close;
- voorzie de custom control klasse van *TemplatePart* attributen die elk van deze drie images benoemen. Als type in het TemplatePart attribuut kiezen we niet *typeof(Image)* maar *typeof(UIElement)* om zo weinig mogelijk beperkingen aan de nodige elementen op te leggen.
  - plaats ook een control die de titel zal bevatten in de Grid, ook al zullen we deze in deze les verder niet verwerken;

- voorzie in de grid van het MainWindow een rij bovenaan waarbinnen de caption control getoond wordt. Verifieer dat de visualisatie klopt met de verwachtingen: indien u het Window resized moet ook de icoontjes resizen!
  - vergeet niet *VerticalAlignment* en *HorizontalAlignment* properties op Stretch te plaatsen en de *Stretch* property van de Images op *Fill*;
  - wellicht zal u de icoontjes die door Blend in de *Themes* folder werden geplaatst moeten verplaatsen naar een *images* folder die u zelf aanmaakt. U moet dan natuurlijk de XAML-verwijzing naar de ico-files ook aanpassen.

Om nu de gewenst acties aan de voorzien Images te koppelen gaan we als volgt te werk:

- override *OnApplyTemplate*. Gebruik *GetTemplateChild* om de UIElementen waaraan logica gekopeld moet worden (deze vermeldt in de *TemplatePart* attributen) op te halen;
  - koppel code aan het *MouseUp* event, wat overeenkomt met een *Click* event voor de gebruiker. Jammer genoeg is *Click* nog niet gekend op *UIElement*, reden waarom we *MouseUp* gebruiken.
  - om een window te sluiten moet u het window waarop de control staat kennen. U kan het window ophalen met een zeer eenvoudige recursieve routine die gebruik maakt van *VisualTreeHelper.GetParent*
    * voorzie een implementatie voor de drie bewerkingen: minimize, maximize en close.
    * u kan het Window waarbinnen een control zich bevind ook opvragen met behulp van een Window.GetWindow- oproep (met dank aan Bram Teirlynck)

Verifieer dat de gevraagde functionaliteit werkt.

9.3.1.3 Caption control via Parts: test en aanpassing

Gebruik Blend om in het formulier nieuwe template code voor de Caption control aan te maken: we vervangen minstens één van de Images door een *Button*. Merkwaardig genoeg zal die knop niet werken, ook al wordt de code doorlopen en het event gekoppeld: omdat sommige control-Types het MouseUp- event zelf afhandelen en *Handled* op *True* plaatsen wordt het event niet meer wordt doorgegeven. We zullen de event-koppeling op een andere manier moeten realiseren:

- kijk na hoe u met behulp van *AddHandler*- instructies een reeds afgehandeld event kan opvangen;
- verifieer dat het gebruik van de knop nu wel het beoogde effect heeft;
- pas ook de code voor de andere controls in de default template aan.

Ter Daels

9.3.1.4 Caption Control via PARTS: evaluatie

- Deze manier van werken levert de client-ontwikkelaar flink wat vrijheid en geeft de control ontwikkelaar de kans om logica te koppelen aan (te verwachten aanwezige) visuele elementen;
- een nadeel is uiteraard de vereiste aanwezigheid van elementen met een specifieke naam, en het moeilijk koppelen van andere controls aan de gewenste logica (veronderstel dat we op de caption twee close- elementen wensen te plaatsen, terwijl er maar één template part hiervoor werd voorzien);
- deze methode is enkel toepasbaar indien u zelf de control ontwikkelt. Indien u een ander event dan dit dat default aan het Command is gebonden wenst op te vangen zal u gebruik moeten maken van de dependency properties benadering die wordt toegelicht in het volgende hoofdstuk.

9.4 Caption Control opgave

- maak een Caption Control zoals hiervoor beschreven en demonstreer zijn werking. Opdat u de images die u via Blend op uw controls plaats ook te zien zou krijgen gaat u best op de volgende manier te werk:
  - maak een *Images*- map in het project op hetzelfde niveau als de *themes*- map. De gemakkelijkste manier om dit te realiseren is rechts klikken in Blend en 'add folder' kiezen;
  - voeg de image- files toe aan deze directory (mbv de verkenner buiten Blend);
  - voeg deze image- files toe aan het project door in Blend rechts te klikken op de Images folder, kies *add existing item* en (multiselect!) kies de verschillende image-files. U zou deze nu ook in Blend moeten zien onder de Images folder;
  - stel deze images als source in voor de controls door deze in de keuzelijst (bij de source property) te kiezen.

met dank aan Sven Meesters (2011-2012)

9.5 States

Een visualisatie heeft hopelijk de mogelijkheid om verschillende statussen (states) van de getoonde informatie weer te geven: is een inputveld fout, teken dan een rode rand. Of zet de tekst zelf in het rood. Of plaats een uitroepteken achter het veld. Of doe een combinatie van voorgaande. Dit voorbeeld geeft aan dat de visualisatie, gekoppeld aan een bepaalde status van de gegevens, best niet bepaald wordt door het software gedeelte dat de fout detecteert, maar wel door het visualisatie gedeelte waarbinnen de fout moet benadrukt worden. We moeten ons natuurlijk niet beperken tot het al dan niet verkeerd zijn van informatie: zo zal een CheckBox het al dan niet aangevinkt zijn aangeven via een state (zie de volgende afbeelding), waarop de

visualisatie beslist hoe dit het best wordt weergegeven.

Figuur 9.3: Checkbox states visualisatie

Om dit gestructureerd mogelijk te maken zullen we als volgt te werk gaan:

- de custom control zal verschillende mogelijke states definiëren via TemplateVisualState-attributen op de control- klasse;
  - in de tekst staat beschreven hoe de onderlinge exclusiviteit gedefinieerd wordt;
  - deze states zijn trouwens ook gekend in Blend;
  - ook de .NET controls werken met dit states- systeem;

Figuur 9.4: Checkbox states via Blend

- in de XAML-code kunnen we de overgang tussen de verschillende states geanimeerd visualiseren:

Jef Daels

```
<Grid x:Name="grd">
    <VisualStateManager.VisualStateGroups>
        <VisualStateGroup x:Name="BezetState">
            <VisualState x:Name="Bezet">
                <Storyboard x:Name="sbBezet">
                    <ColorAnimation Duration="0:0:
                </Storyboard>
            </VisualState>
            <VisualState x:Name="OnBezet">
                <Storyboard>
                    <ColorAnimation Duration="0:0:
                </Storyboard>
            </VisualState>
        </VisualStateGroup>
```

Figuur 9.5: State animaties in XAML

  - u kan deze animaties perfect via Blend aanmaken door een storyboard aan te maken en overgangen tussen states aan te geven;
  - bemerk dat de start- en doel- status niet tot de animatie behoren;
  - indien u de XAML-code zelf codeert: u moet de VisualStateManager tag onmiddellijk onder de eerste control van de visualisatie coderen.

- langs de $C^{\#}$kant van de control moeten de state- overgangen aangegeven worden. Hiervoor moet de VisualStateManager.GotoState methode worden uitgevoerd. De MSDN-topic in de link beschrijft de verschillende parameters. Ook de derde parameter heeft zijn betekenis!
  - dergelijk gedrag leunt zeer sterk aan bij behaviors die later in de cursus aan bod komen.

9.6 State opgave

Ontwikkel een verkeerslicht control volgens het PART- en State model. Het verkeerslicht wordt aangestuurd door op één van de lichten te klikken. Dit wijzigt de state van de control, waardoor een animatie dit licht ontsteekt en de andere lichten dooft (zie demo). Te volgen werkwijze:

- u *moet* werken met parts en states;
- u maakt de animatie om tussen de states te navigeren met behulp van Blend;

Figuur 9.6: State en Parte opgave: verkeerslicht

Jef Daels

Hoofdstuk 10

Behaviors

10.1 Behaviors: duiding

Door middel van dependency properties kunnen events op UIElementen worden afgehandeld. We spreken in dergelijk geval van *behaviors*. Oorspronkelijk geïntroduceerd in Blend 3.0 (en later opgenomen in .NET zelf) kan men behaviors gebruiken om een nieuw *gedrag* te koppelen aan uw UI. Indien de standaard aangeleverde behaviors niet voldoen kan u zelf een behavior ontwikkelen. In dit hoofdstuk zullen we Drag en Resize behaviors implementeren. Wellicht vind u op het internet vele gratis, interessante behaviors, naast deze die reeds in Blend worden aangeboden. Op het einde van dit hoofdstuk wordt een sectie gewijd aan het onthouden van de gewijzigde settings (afmetingen, posities, ..) in de application settings zodat bij het herstarten van de toepassing de aangepaste waardes toegepast worden. Ook een korte toelichting omtrent enumerations komt aan bod.

10.2 Relevante namespaces

- System.Windows.Interactivity

10.3 Drag behavior

Het verslepen van een UIElement is gebaseerd (en aangepast) op WPF tutorial .NET: Behaviors: we wensen de mogeljkheid te ontwikkelen om een control at run-time te verslepen. We zullen dezelfde behavior op drie manieren ontwikkelen:

- door middel van dependency properties: een goeie oefening en hopelijk verduidelijkend;
- via de Behavior die in Blend aanwezig is, maar niet bruikbaar voor Button- objecten;
- door een alternatieve Behavior te ontwikkelen, verder bouwend op de Blend *Behavior*-klasse.

10.3.1 Drag behavior: dependency property versie

We ontwikkelen een dependency property die, indien toegepast op een UIElement, toelaat dit te verslepen. Hiertoe moeten we drie events afhandelen:

- *MouseDownEvent*: we onthouden de huidige locatie (ten opzichte van de parent) en huidige muispositie en onthouden dat een *Drag*-beweging start (bMoveFlag wordt true);
- *MouseMoveEvent*: indien bMoveFlag true berekenen we de afstand die de muis aflegde sinds het MouseDownEvent en gebruiken deze afstand om de locatie van het UIElement met diezelfde afstand te wijzigen (het verplaatsen kan gebeuren door middel van een *TranslateTransform*;
- *MouseUpEvent*: we zetten de bMoveFlag op false, wat de eventuele verplaatsingen stopt.

Een beschrijving van de verschillende nodige elementen vind u in het tweede stuk van WPF tutorial .NET: Behaviors. Gelieve deze implementatie te realiseren en verifieer volgende zaken:

- de code werkt ook wanneer u een UIElement herhaalde keren (in dezelfde sessie) versleept;
- de code werkt ook voor een Button- control.

10.3.2 Drag behavior: Blend out of the box versie

In het voorgaande stukje werd een volledige eigen implementatie verzorgt, gebaseerd op dependency properties. Sinds Blend 3.0 worden een aantal behaviors standaard via Blend aangeboden, waaronder onder andere het verslepen van elementen. Het eerste stuk van hetzelfde artikel (WPF tutorial .NET: Behaviors) leert ons hoe we de Blend behavior *MouseDragElementBehavior* op een UIElement kunnen toepassen:

- volg de werkwijze van dit document en voorzie een control van de *MouseDragElementBehavior*-behavior;
- verifeer dat deze behavior niet werkt op een Button control.

10.3.3 Drag behavior: gespecialiseerde Blend versie

Omdat de standaard Blend-versie niet werkt voor een Button- control zullen we een eigen, Blend gebaseerde versie maken. We zullen hiervoor erven van *Behavior<T>* en de nodige methodes overriden. Het resultaat is een Behavior die ook vanuit Blend instelbaar is op een willekeurig UIElement. We gaan hiertoe als volgt te werk:

- de behavior werkt niet voor een Button, wellicht omdat de Button de klik- events zelf verwerkt en als handled markeert, waardoor ze niet langer *bubblen*. U kan een als *handled=true* bubbling event toch nog opvangen indien u gebruik maakt van de UIElement.AddHandler- instructie;
  - misschien is het ook wel de bedoeling van de Blend drag om niet te werken voor een Button. Immers, het draggen van een Button (mousedown, mousemove, mouseup) lijkt sterk op het klikken op de Button, en kan aanleiding geven tot verwarring;
- maak een nieuwe klasse die erft van *Behavior<UIElement>*. De *Behavior<T>* klasse wordt gevonden in een Blend libary. U kan deze library gemakkelijk terugvinden door middel van het vorige voorbeeld;
- in Behavior Generic Class wordt beschreven welke methodes overschreven kunnen worden. Eén van deze methodes is logisch sterk gelijkaardig aan de value changed handler die werd gecodeerd in de dependency property benadering. We gebruiken deze methode en copiëren de eerder ontwikkelde code in de nieuwe klasse (aangepast waar nodig);
- de nieuwe behavior kan vanuit Blend op een UIElement gesleept worden:
  - u selecteert de behavior (op basis van zijn naam) in het selectiewindow en sleept hem vanuit het selectie window op het UIElement;
  - het is niet mogelijke (in Blend 4) om de nieuwe behavior in de toolbox op te nemen, hij komt wel in de lijst van behaviors terecht.

Verifieer dat de nieuwe oplossing werkt op een Button element.

10.4 Grip behavior

Een *Grip* laat toe om een volledig window in plaats van een individuele control te verslepen. Eerder dan van scratch te herbeginnen en een bijna volledige copy te implementeren zullen we gebruik maken van overerving:

- erf van één van uw eigen drag- implementaties;
- in een volledige WPF omgeving zou het volstaan om de verondersteld aanwezige transformatie niet op het UIElement zelf toe te passen maar op het Window waarbinnen dit UIElement geplaatst werd (te vinden met de *VisualTreeHelper*). Jammer genoeg is een *Window* geen echt WPF object en kan men hierop geen RenderTransform instellen. We kunnen het window verplaatsen door zijn Top en Left properties aan te passen;

  - override de mousemove handler;
  - bereken, net zoals in de parent klasse de af te leggen afstand;
  - gebruik de *Top* en *Left* properties van het window om dit te verplaatsen;
  - het kan natuurlijk nodig zijn om een aantal elementen in de parent klasse niet langer als private maar nu als *virtual* te declareren.

10.5 Resize behavior

Een gelijkaardige (iets moeilijker) behavior is een *Resize*- behavior: het is de bedoeling om, wanneer op de rand van een control wordt geklikt, deze kan geresized worden. Hiervoor gaan we als volgt te werk:

- we erven van de Blend- Behavior classe en koppelen drie events;
- op het *MouseDownEvent* zullen we nakijken of er aan de rand wordt geklikt. Zoja wordt de cursor van het UIElement gewijzigd en onthouden we de mousedown positie;
- op het *MouseUpEvent* herzetten we de cursor tot zijn default;
- in het mousemove event verwerken we de verplaatsing op volgende wijze;
  - indien u moet resizen naar het oosten of naar onder volstaat het om de afmeting van de control aan te passen en de huidige muispositie als nieuwe mousedown te registereren;
    * indien u de mousedown positie niet vervangt wordt de afstand telkens ten opzicht van de originele positie genomen, zodat eerdere bewegingen terug geteld worden.
  - indien u naar het westen of noorden beweegt moet u de afmetingen van de control aanpassen *en* de control verplaatsen. Normaal gezien moet u de nieuwe muispositie niet registeren omdat die door het verplaatsen van de control terug samenvalt met de originele mousedown positie
    * net zoals in de Drag behavior oplossing is het verplaatsen van de control het gemakkelijkst te realiseren door middel van een translate transform.

Omdat we de *Width* en *Height* van een control manipuleren zal deze implementatie slechts werken indien deze reeds werden ingesteld. In een professionele implementatie is het nodig om, indien de Width of Height nog niet werden ingesteld, gebruik te maken van respectievelijk *ActualWidth* en *ActualHeight* om de berekeningen uit te voeren.

10.5.1 Alignment impact (Laurenz Demey 2010- 2011)

Wanneer de te resizen control *right aligned* is werkt bovenstaande code niet: de speciale afhandeling voor een resizen naar links (verbreden en verplaatsen) moet dan toegepast worden op de rechterkant (en mag niet toegepast worden op de linkerkant (analoog voor onder en bovenkant)

Jet Daels

10.6 Behavior opgaves

- implementeer en demonstreer de Drag behavior als dependency property;
- illustreer de werking van de Drag behavior gevonden in Blend;
- implementeer en demonstreer de Drag behavior als ervend van *Behavior*<*T*>;
- implementeer een *Grip* behavior. Die laat toe om niet een control maar een volledig Window te verplaatsen. Illustreer de werking door deze toe te passen op de Caption controls gemaakt in Templated Controls;
- implementeer een resize behavior;
- plaats de Grip behavior op de Caption Control;
- aanrader: maak een Caption Control zoals deze in het vorige hoofdstuk, maar koppel de logica aan de UI-elementen door middel van dependency properties.
- een model examen opgave had kunnen zijn: pas de F1URL implementatie aan zodat het een behavior wordt:
  - u bent dan niet langer gebonden aan de eigen grid klasse waarbinnen de controls moeten geplaatst worden;
  - u hebt niet langer nood aan een hittest routine om te bepalen welke control de help opvraagt;
- een model examen opgave zou kunnen zijn: ontwikkel de *Behavior*- klasse zelf voor de Windows Store omgeving;

10.7 WPF en application settings

Een interessante feature in de WinForms-designer die ontbreekt in WPF-toepassingen is het gebruik van application settings (instellingen) die UI-properties over sessies heen onthouden. Mits een weinig infrastructuur werken loopt dit echter wel los:

- voeg in Visual Studio aan de properties van uw toepassing de nodige settings manueel toe (kies een geschikte naam en datatype). De designer zal een application settings aanmaken met een static default instantie (u kan de code nakijken in de file *Settings.Designer.cs*.
- koppel de te onthouden WPF eigenschappen aan hun overeenkomstige setting:

```
Height="{Binding Source={x:Static src:Properties.Settings.Default}, Path=hoogte, Mode=TwoWay}"
```

Figuur 10.1: Height en applicationsettings

- anders dan in *VB.NET* worden in *C#*de application settings niet automatisch bewaard. Daarom zullen we in het *Closed* event van het Window de applicationsettings zelf bewaren: *JDADragBehavior.Properties.Settings.Default.Save();* (waarbij JDADragBehavior de naam van het project is).

Jef Daels

10.8 Enumerations: getalwaardes en bewerkingen

In mijn eigen implementatie van de Resize behavior ontwikkelde ik en enumearation om de rand waarbij de cursor zich bevind bij te houden:

```
public enum RandPositie
    {
    Midden=0,
    Noord=1,
    Zuid=2,
    Oost=4,
    West=8
    }
```

Figuur 10.2: Enumeration definitie

Indien een variabele een combinatie van verschillende waardes moet bevatten (stel Noord en West) dan worden beide waades opgeteld met de — operator. Indien we willen testen of een waarde onder andere de Noord waarde bevat gebruiken we de *&* operator. Door de getalwaardes als machten van 2 te kiezen is elk getal slechts op exact één manier een combinatie van de voorgaande. Hierdoor is geen verwarring mogelijk.

- het getal 7 is met voorgaande definitie slechts mogelijk door combinatie van 1 (Noord), 2 (Zuid) en 3 (Oost);
- 7 & Zuid levert een waarde verschillend van 0 (midden);
- 7 & West lever 0 (midden) op.

Om de uniciteit van de combinaties te begrijpen kan een binaire invalshoek helpen:

- doordat de getallen machten van 2 zijn, zijn ze in een binaire voorstelling allemaal 1 gevolgd door een aantal nullen. Het combineren van verschillende enum-elementen zet de overeenkomstige 1-waardes actief (elke 1-waarde kan slechts door 1 enum waarde worden opgezet, gezien we geen dubbels toelaten). Het getal resultaat wordt voor het leesgemak in de code decimaal genoteerd;
- indien u de binaire voorstelling van een getal nakijkt kan u op basis van de 1-tjes nakijken welke enumwaardes deel uitmaken van dit getal. De & operator is uitermate geschikt om na te kijken of een specifieke enumwaarde (die maar één 1 bevat) deel uitmaakt van een gecombineerde set waardes.

10.9 Extra lectuur

- The Attached Behavior pattern: een theoretische duiding;
- Introduction to Attached Behaviors in WPF: een zeer eenvoudig en verklarend voorbeeld;
- WPF tutorial .NET: Behaviors: een mooi voorbeeld in twintig regels code, verifieer of het werkt bij herhaalde uitvoeringen van dezelfde sessie;
- Behavior Generic Class

Jef Daels

Hoofdstuk 11

iTextSharp

iText is een in België onwikkelde (open source) bibliotheek waarmee PDF-documenten gegenereerd en gemanipuleerd kunnen worden. Deze bibliotheek gebruik ik bijvoorbeeld om een watermerk in deze PDF te plaatsen.

11.1 Installatie

- start een nieuw WPF project;
- *Project.Manage NuGet Packages ..*;
- zoek op *iText* en installeer *iTextSharp* (de $C^{\#}$versie van iText);

11.2 PDF creatie (en opgaves)

Voor een eenvoudige tutorial omtrent het aanmaken van een nieuwe PDF verwijs ik graag naar Basic PDF Creation Using iTextSharp.

Een beetje rondzoeken op het internet helpt u wel verder omtrent een aantal topics. Zo moet u in uw test document volgende zaken realiseren:

- plaats een figuur in de pdf en herschaal deze tot 150x150;
- maak een titel paragraaf met een grotere fontsize en gecentreerd;
- maak een paragraaf die bestaat uit chunks die elk hun eigen kleur hebben;
- trek een lijn in een pdf. Let wel: het PDF coördinaten systeem is verschillend van dit dat u gewoon bent in een informatica omgeving:
  - de oorsprong is links onder op het blad, en niet links boven;
  - een grotere y- waarde ligt hoger op het blad, terwijl dit in een windows omgeving lager ligt.
- ...

11.3 Manipulatie van een bestaande PDF (en opgaves)

- maak een nieuw PDF document door een aantal PDF-documenten achter elkaar te plaatsen;
- lees een pdf document en plaats 4 logische pagina's op 1 fysisch blad;
- plaats een watermerk op elke even pagina van een document.

11.4 Tekenen (en opgaves)

Voorzie uw Sudoku oplossing van de mogelijkheid om een Sudoku te exporteren naar PDF:

Figuur 11.1: Sudoku in PDF

Let op volgende zaken:

- de opgave- getallen zijn anders gekleurd dan de oplossing- getallen;
- de 3x3- hokjes zijn van elkaar gescheiden door dikkere lijnen;

11.5 Toepassingsgebied voor iText

- *niet* als tekstverwerker;
- wel als rapport generator (veel veel sneller dan office activeX);

Jef Daels

Hoofdstuk 12

SQLServer en .NET integratie

Relationele database laten meestal toe om logica in de vorm van stored procedures in de database op te nemen. Naast administratieve voordelen betreffende security en onderhoud is ook de potentiële performance winst een troef: berekeningen op de server dienen niet alle nodige gegevens over het netwerk naar de client te sturen, enkel het resultaat sturen is voldoende. Oorspronkelijk diende deze logica in een RDBMS specifieke, behoorlijk ouderwetse taal gedefinieerd te worden. Sinds enige jaren bestaat de mogelijkheid om de logica in uw favoriete programmeeromgeving te ontwikkelen en deze daarna in de server te plaatsen. Dit hoofdstuk behandelt het opnemen van .NET programmatuur in SQLServer.

12.1 Nodige softwares

- Visual Studio 2012;
- SQL Server 2012;
- SQL Server Datatools 2012.

Installatie van de SQL Server Datatools maakt een nieuwe Visual Studio versie beschikbaar. Deze dient u te gebruiken voor deze les.

12.2 Kwadraten berekenen

Wellicht weinig zinvol in een database context, maar gemakkelijk om het principe uit te leggen is het aanmaken van een functie die bij oproep een tabel van kwadraten aanlevert. Indien u zelf meer voorbeelden wil zal Usage Scenarios and Examples for Common Language Runtime (CLR) Integration wellicht een goed startpunt zijn.

- start *SQL Server Data Tools 2012*, een *Visual Studio* oplossing;
- voeg aan de SQL Server object explorer uw lokale SQL Server 2012 installatie toe;
- voeg de database SQLNET toe. Dit is de database waarbinnen we onze test zullen uitvoeren;
- lees mij;
- *Create New Project* op de zonet gemaakte database, met als naam: *KwadraatDemo*;
- voeg de klasse *Kwadraat* toe:

```
public class Kwadraat
    {
    private SqlInt64 _getal, _kwadraat;
    public Kwadraat(SqlInt64 getal, SqlInt64 kwadraat)
        {
        _getal = getal; _kwadraat = kwadraat;
        }
    [SqlFunction(FillRowMethodName = "VentileerRijen", TableDefinition = "getal bigint, kwadraat bigint")]
    public static IEnumerable MaakKwadraten(int maxgetal)
        {
        try
            {
            ArrayList kwadraten = new ArrayList();
            for (int i=1;i<=maxgetal;i++)
                kwadraten.Add(new Kwadraat(i, i*i));

            return kwadraten;
            }
        catch (Exception ex) { return null; }
        }
    public static void VentileerRijen(object okwad, out SqlInt64 getal, out SqlInt64 kwadraat)
        {
        Kwadraat kwad = okwad as Kwadraat;
        getal = kwad._getal;
        kwadraat = kwad._kwadraat;
        }
```

Figuur 12.1: Kwadraat code

- compileer en publish deze code;
  - voeg de nodige referenties aan het project toe;
  - bemerk dat SQLServer datatypes verschillend zijn van .NET datatypes;

Jef Daels

  - bij het publishen moet u de nodige parameters voorzien om in de juiste database, op de juiste server te deployen;

- indien het publishen lukt zal u in de database onder de *table value functions* de routine *MaakKwadraten* terug vinden.

- u kan deze functie gebruiken om een tabel kwadraten te selecteren:

Figuur 12.2: Kwadraten selecteren

Omtrent bovenstaande code verwijs ik graag naar het internet voor een eerste toetsting van uw vragen.

12.3 SQL Server tabellen raadplegen

Wanneer u in een klassieke toepassing SQL Server tabellen raadpleegt dient u een database connectie te configureren. Omdat we nu code ontwikkelen die in SQL Server opgenomen zijn en dus geconsulteerd worden via een bestaande connectie die we willen hergebruiken zal dit op een net iets andere manier gerealiseerd worden

- maak een tabel (vergeet de PK niet) in de SQLNET database en voorzie deze van een aantal rijen;

- lees mij;

- gebruik uw verzamelde kennis (deze les en vorige jaren) om een TVF te maken die iets uit die tabel ophaalt en terug geeft. Maak bijvoorbeeld een select met een where- conditie waarbij een parameter wordt meegegeven (zie demo).

De kans is groot dat dit niet onmiddellijk lukt, wat ons onmiddellijk de nood aan debugging demonstreert.

12.4 Debugging SQLServer

12.4.1 Debugging startend vanuit het Visual Studio project

Het opzetten van de debugging in SQLServer 2012 gebeurt net iets anders dan in de vorige SQL Server versies:

- in de *SQL Server Object Explorer* (dit is *niet* SQL Server Explorer) kan u door te rechtsklikken het debuggen aanzetten;
- in de properties van het project verifieert u in de *Debug* farde of de connection settings in orde zijn. Gebruik de edit- knop indien dit niet het geval zou zijn;
- voeg een script toe aan het project waarin u het te testen SQL- statement opneemt. Opnieuw in de debug farde stelt u het startup script in op dit script (er staat een browse knop helemaal rechts);
- plaats break points in uw .NET code en start het project. Indien alles goed verloopt komt u in een debug sessie terecht.

Het debuggen, interpreteren en oplossen is voor de lezer.

12.4.2 Debugging van een SQL Server call gestart buiten het project

Indien bijvoorbeeld in Management Studio software een query gestart wordt die code uitvoert waarvan wij het project hebben kunnen we deze uitvoering als volgt debuggen:

- open het Visual Studio project en attach aan *sqlservr.exe*;
  - *Tools.Attach to process ...*;
  - zet *show processes from all users* op omdat de sqlserver service wellicht niet met uw eigen userid gestart is;
  - bij het attachen krijgt u een beveiligings boodschap die u waarschuwt voor eventuele risico's;
  - dit attachen brengt uw project trouwens in een running state;
- voer een select statement uit die de code in uw Visual Studio project uitvoert. Indien u een breakpoint plaatste zal de uitvoering daar onderbroken worden en kan u uw gebruikelijke debug acties starten.

12.5 Theorie vragen

Met onderstaande vragen test u uw begrijpen van de software en krijgt u een idee omtrent de mogelijke detailvragen voor dit hoofdstuk.

- het SqlFunction- attribuut heeft een FillRowMethodName parameter. Deze wordt in onze klasse niet opgeroepen. Leg uit wat de functie van deze methode is.
- het SqlFunction- attribuut heeft een TableDefinition parameter. Welke zijn de mogelijke datatypes?
- de datatypes gebruikt in de TableDefinition parameter zijn niet identiek aan deze gebruikt in de FillRowMethode. Verklaar!
- opdat .NET programma's in SQLServer bruikbaar zouden zijn moet een setting worden ingesteld. Is deze setting een server, database of assembly instelling?
- opdat uw .NET programma's in SQLServer via Visual Studio debugbaar zouden zijn moet een setting worden ingesteld. Is deze setting een server, database of assembly instelling?

Jef Daels

Hoofdstuk 13

Windows installer oplossingen

Om een desktop toepassing te verspreiden is het wenselijk om gebruik te maken van een MSI- bestand (Store toepassingen worden verspreid via de Store). Dergelijk installatiebestand ondersteunt een aantal standaard-acties zoals daar zijn: het voorzien van desktop- en start-icoontjes, het voorzien van een *uninstall* mogelijkheid, het automatisch mee- installeren van de nodige dll's en dergelijke meer.

Vanaf versie 2012 van Visual Studio verdwijnt de mogelijkheid om *Visual Studio setup-projecten* in een solution op te nemen. Deze feature werd vervangen door de introductie van *InstallShield Limited Edition for Visual Studio*, de gratis variant van de klassieker InstallShield. Het is belangrijk op te merken dat er ook andere al dan niet gratis softwares bestaan waarmee *MSI* pakketten kunnen ontwikkeld worden. In deze cursus komen het reeds vermeldde *InstallShield Limited Edition* en WiX (ontwikkeld door Microsoft) aan bod.

13.1 Demo programma

Ontwikkel een WPF-toepassing met volgende features:

- er worden twee assemblies aangeleverd: een .exe en een .dll;
- de UI van de .exe toont een knop die, wanneer hij wordt geklikt, een string uit de .dll ophaalt en toont in een MessageBox.
- we hebben een .ico file die we als desktop icoon wensen te gebruiken voor deze toepassing.

13.2 InstallShield Limited Edition

13.2.1 Download en installatie

Even Googlen en u vindt een gepaste download. Het resultaat van de installatie integreert netjes met Visual Studio waar een nieuw project type beschikbaar wordt gesteld onder *Other Project Types/Setup And Deployment*.

13.2.2 Aanmaken van een MSI-pakket

Het is de bedoeling dat de student zelf wat experimenteert met de software. U dient minstens volgende zaken te realiseren:

- maak een setup waarin minstens twee files voorkomen die geïnstalleerd moeten worden;
- wijzig de schermen van het installatie proces: gebruik images en wijzig de licentie tekst;
- de installatie moet een desktop-icon en een startmenu icon voorzien (het startmenu icon werkt momenteel niet onder windows 8).

13.2.3 Redistributables

Jef Daels

Wellicht is uw software afhankelijk van andere softwares, zoals het .NET framework, SQLServer, .. . U kan deze softwares meenemen in het installatiepakket. Om één of andere reden lukt dit me niet voor het .NET framework bij het gebruik van de gratis InstallShield.

13.2.4 Evaluatie InstallShield Limited Edition

Ik ben er niet wild van:

- na één jaar Windows 8 is het geïnstalleerde programma nog steeds niet op het start scherm te plaatsen;
- redistributables opnemen in het pakket (of deze automatisch door de client laten downloaden) lukt me niet. Misschien enkel in de betalende versies?
- een groot aantal features wordt in de gratis versie enkel vermeld maar kan pas gebruikt worden in de betalende versies:
  - het bepalen van dependencies gebeurt niet in de gratis versie;
  - setup- design;
  - ...

Deze software lijkt me bruikbaar in eenvoudige situaties, zoals een lescontext. Een voordeel is dan weer wel dat de wizard driven interface gemakkelijk in gebruik is. Wellicht is de betalende versie ook bruikbaar in complexere situaties.

13.3 WiX

Wix is een XML gestuurde software met een stijle leercurve. Een tutorial is dan wellicht ook op zijn plaats.

13.3.1 Download en installatie

Google .. .

13.3.2 Aanmaken van een MSI-pakket

Het is de bedoeling dat de student zelf wat experimenteert met de software. U dient minstens volgende zaken te realiseren:

- maak een setup waarin minstens twee files voorkomen die geïnstalleerd moeten worden;
- wijzig de schermen van het installatie proces: gebruik images en wijzig de licentie tekst;
- de installatie moet een desktop-icon en een startmenu icon voorzien (het startmenu icon werkt momenteel niet onder windows 8).

13.3.3 Redistributables

Wellicht is uw software afhankelijk van andere softwares, zoals het .NET framework, SQLServer, .. . U kan deze softwares meenemen in het installatiepakket. In een Wix- benadering moet u een extra *Bootstrapper Project* (dit is een speciaal project type) aanmaken. Dit project type specifieert een *Chain* van te installeren softwares. Uw eigen software (via het WiX installatie project) en de te installeren software (bijvoorbeeld het .NET framework) maken hier onderdeel van uit.

13.3.4 Evaluatie WiX

De leercurve van WiX is *veel* stijler dan deze van InstallShield. Toch verkies ik Wix boven InstallShield (een privé mening), wellicht om volgende redenen:

- de gratis versie is geen beperkte versie;
- de (complexe) XML- aansturing (en de online documentatie) geven me toch wel het idee dat ik de meeste situaties geconfigureerd kan krijgen, waar ik met de gratis InstallShield versie toch wel wat beperkingen wordt opgelegd.

13.4 Welke installer?

13.4.1 Vergelijking InstallShield en Wix

Indien geld geen probleem zou zijn verwacht ik een grote meerwaarde bij het gebruik van een betaalde InstallShield installatie. Indien enkel de gratis versies in overweging worden genomen verkies ik WiX omdat ik vrees dat de free InstallShield versie niet voor alles toereikend zal zijn, en ik op termijn dus toch WiX zal nodig hebben.

13.4.2 Vergeet niet:

Er zijn nog andere installatie softwares, zowel gratis als betalend, dit hoofdstuk heeft geenszins de bedoeling om een uitputtende vergelijking tussen de verschillende mogelijkheden te maken.

13.5 Voorbeeld labo examen opgave

- maak een setup voor uw sudoku programma, zowel met InstallShield als Wix;
- voorzie in de setup ook een PDF met een getekende Sudoko die mee geïnstalleerd wordt (de veronderstelde help).

Jef Daels

Hoofdstuk 14

AOP: Aspect Oriented programming

14.1 AOP duiding

Bij het ons welbekende programmeren is het gebruikelijk om programmalogica logisch op te delen in verschillende klasses en methodes: elke methode staat in voor de correcte afhandeling van een deelprobleem. Hoewel dit suggereert dat elk probleem op één welbepaalde plaats wordt afgehandeld zijn er *aspecten* van het programma die zich *niet* tot een welbepaalde plaats beperken: *logging*, *parameter validatie*, *authorisatie checks*, ... komen telkens opnieuw, op verschillende plaatsen terug. *Aspect Oriented Programming* biedt de mogelijkheid om de aspect gerelateerde code op één welbepaalde plaats te coderen:

.NET ondersteunt AOP door middel van de ContextBoundObject- klasse. Omwille van de beperktheid van deze oplossing zullen we voor deze lessen gebruik maken van *PostSharp*, hoewel dit natuurlijk niet het enige alternatief is. In Rating of Open Source AOP Frameworks in .NET worden een aantal frameworks met elkaar vergeleken. Recent werd *PostSharp* een commercieel produkt (het was oorspronkelijk gratis), reden waarom we gebruik maken van een trial versie.

Onderstaande afbeelding illustreert het beperken van een examenpunt tot een getal tussen nul en twintig, door middel van een attribuut. Het afdwingen van de parameter- beperking zal gebeuren door middel van het *OnEntry*- aspect.

```
public static void ingaveStudentModulePunt(string student, string module,[AOPIntegerRange(0,20)] int punt)
    {
    ////parameters testen
    //if (punt < 0) throw new Exception(punt + " is kleiner dan 0");
    //if (punt > 20) throw new Exception(punt + " is groter dan 20");

    Console.WriteLine(student + " haalde voor " + module + " " + punt + " punten.");
    }
```

Figuur 14.1: AOP: parameter annotatie

14.1.1 Relevante namespaces

- PostSharp.

14.2 PostSharp

14.2.1 PostSharp kennismaking: methode-entry teller

Indien u versie 2.5 installeert (Google) gaat u na installatie als volgt te werk:

- ontwikkel een Fibonacci oplossing
- use *Project.Manage NuGet packages* to add the *Postsharp-Aspects* package;
- maak een eigen klasse *clsEntryTeller* die erft van *PostSharp.Aspects.OnMethodBoundaryAspect*;
- override de *OnEntry* methode. Deze wordt uitgevoerd telkens een methode, voorzien van het *clsEntryTeller* attribuut wordt gestart (we zullen dit attribuut op alle methodes van een klasse toepassen door het attribuut op klasse niveau te definiëren);
- voorzie logica om de uitvoering van deze methode te registreren. Dit zal wellicht gebeuren door middel van een static field in *clsEntryTeller* dat voor elke methode het aantal uitvoering bijhoudt (gebruik een *Dictionary*?);
- voorzie logica (statisch?) om de informatie, verzameld in *clsEntryTeller*, in de console te dumpen.
- programmeer wat test- logica en voorzie de te tellen methodes van het *clsEntryTeller* attribuut. Dit kan gebeuren op verschillende manieren:
  - u definieert het attribuut voor de methodehoofding (bruikbaar voor een klein aantal methodehoofdingen);
  - u definieert het attribuut op klasse (of assembly)- niveau en vermeld (met behulp van reguliere expressies) welke methodes al dan niet van het attribuut moeten voorzien worden.
- test het resultaat.

PostSharp weeft IL-code rond de methodeoproepen waarop het *OnMethodBoundaryAspect* werd toegepast. De door PostSharp geïntroduceerde namen bevatten tekens die door ons bij het programmeren niet gebruikt kunnen worden. Dit vermijdt potentiële naamconflicten tussen onze code en de gegenereerde code. Zoals reeds werd vermeld in het hoofdstuk omtrent runtime compilatie wordt dit code-weven gestart door in het *MSBuild* proces in te grijpen.

Jor Daels

ILSpy maakt duidelijk hoe de uiteindelijke gegenereerde IL code er uit ziet (voor de hieronder gegenereerde IL-code werd ook de *OnException* methode overriden, zodat het hier getoonde try-catch blok werd gegenereerd):

```
public class Wiskunde
{
    [System.Diagnostics.DebuggerNonUserCode, System.Runtime.CompilerServices.CompilerGenerated]
    internal sealed class <>z__Aspects...
    public static int Fibonacci(int i)
    {
        MethodExecutionArgs methodExecutionArgs = new MethodExecutionArgs(null, null);
        MethodExecutionArgs arg_13_0 = methodExecutionArgs;
        System.Reflection.MethodBase m = Wiskunde.<>z__Aspects.m3;
        arg_13_0.Method = m;
        Wiskunde.<>z__Aspects.a0.OnEntry(methodExecutionArgs);
        int result;
        try
        {
            int num;
            if (i <= 0)
            {
                num = 0;
            }
            else
            {
                if (i == 1)
                {
                    num = 1;
                }
                else
                {
                    num = Wiskunde.Fibonacci(i - 1) + Wiskunde.Fibonacci(i - 2);
                }
            }
            result = num;
        }
        catch (System.Exception CS$0$5__exception_58)
        {
            Wiskunde.<>z__Aspects.a0.OnException(methodExecutionArgs);
            throw;
        }
```

Figuur 14.2: PostSharp en ILSpy

14.2.2 PostSharp en parameter annotaties

Het codevoorbeeld in de duiding illustreert de opzet: we wensen *précondities* aan formele parameters op te leggen door middel van attributen. In het hieronder uitgewerkte voorbeeld zal een *préconditie* enkel van toepassing zijn op een individuele parameter. We wensen de volgende situtatie te bekomen:

- bij de start van een methode zal het *OnEntry* aspect de eventueel aanwezige parameter-attributen (te vinden in de formele methode declaratie) checken;
- elk préconditie attribuut verifieert of de actuele parameter waarde voldoet. Indien niet wordt er een fout gegooid;
  - we leggen aan elk préconditie attribuut de interface *IAOPValidatie* op. Op deze manier is het gemakkelijk na te gaan of een parameter attribuut door onze infrastructuur moet gechecked worden;
  - we voorzien die interface van een *valideer* methode. Op die manier kan elk attribuut op dezelfde manier verwerkt worden (zodat we met een eenvoudige lus alle attributen kunnen overlopen en verwerken).

Meer in detail gaan we voor de attributen als volgt te werk

- definieer de interface *IAOPValideer* en voorzie de *valideer*- methode. Het return type is *void* en de routine heeft één parameter van type *object* (op deze manier kan het valideren van een datum of string door dezelfde virtuele methode opgevangen worden;
- definieer préconditie klasses die deze interfaces implementeren. De constructor (gebruikt in de attribuut definitie van de formele parameter) neemt welilcht één of meerdere parameters (range- condities nemen wellicht twee argumenten in de constructor);
- implementeer de *valideer*- methode. Deze throwt een *AOPValidatieException* indien de voorwaarde niet voldaan is.

Voor het aspectorientatie gedeelte voorzien we volgende code:

- voorzie een (*PostSharp*) *OnMethodBoundaryAspect*;
- override de *OnEntry*- methode. Via *args.Method.GetParameters* bekomt u de formele parameters van de oproep (*args* bevat de actuele parameters). De formele en actuele parameters stemmen positioneel met elkaar overeen (de eerste actuele parameter (postie 0) stemt overeen met de eerste formele parameter (ook positie 0)).
- schrijf wat code om het voldaan zijn van de eventuele attributen te checken (twee lussen, vijf lijntjes logica code (dus zonder accolades of routine hoofdingen))!

Houd ook rekening met volgende zaken:

- wanneer u uw code test, beperk u dan niet tot précondities die niet voldaan zijn. Indien de préconditie wel voldaan is moet het programma uiteraard verder werken;
- plaats eens meerdere précondities op hetzelfde argument;
- in de *PostSharp* documentatie vinden we dat het construeren van de method- informatie een inefficient proces is. De hierboven geschetste implementatie zal deze informatie bij elke methode oproep verzamelen. U kan dit beperken door de methode informatie te verzamelen in de *RuntimeInitialize* uitvoering, en deze dan op module niveau (voor deze specifieke methode, dus niet *statisch*) te bewaren.

Hopelijk realiseert u zich dat de attributen die worden geplaatst bij de formele parameter definities op zich geen AOP introduceren: we plaatsen met deze attributen meta-informatie op de parameters, die dan door de AOP-verwerking (de method- entry) gebruikt worden om de extra logica (de aspect logica) te sturen.

14.2.3 PostSharp en performance counters

Performance Counters zijn (sinds NT) *de* manier om prestatie cijfers omtrent uw toepassing aan het operating systeem mede te delen. Third party toepassingen kunnen deze counters lezen en visualiseren (of rapporteren):

Figuur 14.3: PDFStamper performance counter

Bovenstaande figuur toont de performancemonitor resultaten van mijn PDFWatermark toepassing (op elke bladzijde wordt een watermerk geplaatst). U merkt duidelijk dat er twee uitvoeringen van het programma voorkwamen. De hier getoonde performance monitor zit standaard in Windows (de screenshot is genomen in een Vista- installatie).

Windows en .NET leveren reeds een grote lijst performance counters. U kan deze gemakkelijk nakijken vanuit de *Server Explorer* in *Visual Studio*:

Figuur 14.4: Performance counters in Visual Studio

Bij een eerste test zal u wellicht geïnteresseerd zijn in hardware gerelateerde counters zoals de *batterijstatus* of *processor* gegevens.

14.2.3.1 Eigen performance counters

Voor het publiceren van eigen performance counters verwijs ik naar How do I... Publish a custom application performance counter in $C^{\#}$?. U zal dit uiteraard moeten beheersen vooraleer u uw eigen performance counters via AOP kan aansturen (het PerformanceCounterType is zeer belangrijk).

14.2.3.2 AOP en performance counters

De performance counters worden vooral gebruikt om toepassingen te profileren: hoe gedraagt een toepassing zich tijdens haar uitvoering? Performance counters kunnen dan ook best gebruikt worden om betekenisvolle activiteiten van uw toepassing te publiceren. Denken we hierbij bijvoorbeeld aan: het aantal vewerkte klanten (per seconde), het aantal failures (per seconde), .. . Om dit met AOP te ondersteunen denken we aan volgende implementatie:

- we wensen een performance attribuut op methode niveau toe te passen, en de mogelijkheid te hebben om op *OnEntry*, *OnException* en *OnSuccess* in te grijpen. *OnMethodBoundaryAspect* is hiervoor uitermate geschikt;
- op basis van de gegevens, nodig om een goede performance counter te definiëren, zullen we bij het overerven extra parameters aan het attribuut toevoegen;
- gebruik deze extra info om op de gewenste plaatsen een performance counter aan te sturen: u bent natuurlijk *niet* verplicht om zowel het falen als slagen van een methode uitvoering te registreren.

14.3 AOP oefeningen

- definieer een attribuut, plaatsbaar op een formele parameter, dat gecombineerd met AOP (ook zelf te maken) verifieert dat een integer waarde zich tussen een onder- en bovengrens bevindt;
- definieer een attribuut, plaatsbaar op een formele parameter, dat gecombineerd met AOP (ook zelf te maken) verifieert dat een geboortedatum een reeds geboren persoon (datum niet in de toekomst) van maximaal 120 jaar definieert;
- definieer een AOP-attribuut, plaatsbaar op een methode, waarmee wordt afgedwongen dat de uitvoerder van een routine een administrator op het toestel moet zijn;
- definieer een performance counter die met behulp van AOP op methodes kan worden geplaatst. De counter telt het aantal uitvoeringen van een routine. Gebruik deze om het aantal uitvoeringen van een aantal van de recursieve methode oproepen uit de voorbije lessen te tellen:
  - tel het aantal recursieve oproepen in uw fibonacci- oefening;
  - tel het aantal recursieve oproepen in uw backtrack- oefeningen (N-Queens en Sudoku)
  - tel het aantal PDF pagina's dat geschreven wordt met uw PDF- merge oefening.

  Het resultaat van deze performance counters moet in de performance monitor zichtbaar zijn

Jef Daels

Hoofdstuk 15

Query Notification

Query Notification biedt de mogelijkheid om gegevens wijzigingen in de database via een event in uw client programa op te vangen. Deze techniek vooral zinvol wanneer u weinig veranderende gegevens cacht in uw client (het bijhouden van lijsten informatie).

Het opzetten van *Query Notifications* is niet zo moeilijk maar vergt enig lees- en opzet werk.

15.1 Query notification oefening

Demonstreer het kunnen werken met Query Notification.

Jef Daels

Hoofdstuk 16

Discussie topics

16.1 Duiding

Als ontwikkelaar moet u in team kunnen werken. In dit team zijn wellicht ook andere ontwikkelaars aanwezig. De hieronder beschreven discussie topics worden door de klas besproken tot een consensus omtrent oplossing wordt bekomen:

- er wordt slechts één algoritme per klas per topic aanvaard;
- iedereen implementeert dit algoritme tegen de volgende les (absentie systeem);
- de week erna volgt een nabespreking, die wellicht resulteert in een herdiscussie;
- de begeleidende docent probeert de discussie *niet* te sturen.

Naast de oplossing voor de technische problemen zijn vooral volgende elementen van belang:

- ook al bent u overtuigd van het werken van uw oplossing, toch moet u ook de oplossing van iemand anders kunnen aanhoren:
  - welke punten uit zijn oplossing zijn een verbetering voor mijn eigen oplossing?
  - welke punten uit zijn oplossing zijn minder goed dan mijn eigen oplossing, en hoe kan ik dit beargumenteren
- in een werksituatie is het niet genoeg om gelijk te hebben: u moet de collega's overtuigen van uw gelijk (vermijd het Cassandra syndroom)
  - hoe krijg ik een complexe redenering uitgelegd? gebruik ik voorbeelden? terminologie?
  - hoe beschrijf ik het foutieve in een redenering?

16.2 Sinus berekenen mbv Taylor reeks

Het berekenen van een sinus kan gebeuren door middel van een Taylor reeks. U vindt een beschrijving hiervan op Wikipedia (Taylor reeks).

Groepsdiscussie: bespreek de te implementeren oplossing.

Jef Daels

Hoofdstuk 17

Global Assembly Cache: GAC

17.1 Global Assembly Cache: duiding

.NET herintroduceerde *copy deployment*, reeds bekend vanuit het *DOS*-tijdperk. Copy deployment bestaat erin dat de installatie directory van de ene naar de andere computer wordt gecopieerd, waarna dit programma ook op de tweede computer werkt.

In een tijd toen disk-space duur was (het DOS tijdperk) introduceerde dit veel overtollige copiën van dezelfde, door iedereen gebruikte dll's. Ook is het niet altijd gemakkelijk voor een programma om het reeds geïnstalleerd zijn van een ander programma te achterhalen: in theorie moet je de ganse schijf doorzoeken. Om aan deze nadelen tegemoet te komen ontwikkelde Microsoft de *registry*: elk programma dat zich installeerde registreerde zichzelf en zijn onderdelen in de registry. Nieuwe versies werden over bestaande versies heen gecopieerd (hun locatie was te vinden in de registry), en programma's vonden elkaar terug door de registry te ondervragen.

Dit gebruik van de registry resulteerde in tal van problemen, in detail beschreven in DLL hell(Wikipedia):

- indien een nieuwe versie niet 100% backwards compatibel is zal een programma dat vroeger werkte dit potentieel niet meer doen: het is perfect mogelijk om twee programma's te ontwikkelen die niet op hetzelfde toestel geïnstalleerd kunnen worden;
- het installeren van twee versies van hetzelfde programma is quasi onmogelijk;

.NET herintroduceert copy-deployment: elke toepassing komt met zijn eigen set dll's in zijn installatie directory: er is geen versie conflict meer. Voor systeem routines is dit een moeilijk houdbare zaak: sommige dll's zullen we toch centraal voor alle toepassingen wensen te installeren. Dit gebeurt in de Global Assembly Cache.

17.1.1 Relevante namespaces

17.2 GAC: technische duiding

pputer ▸ Local Disk (C:) ▸ Windows ▸ assembly ▸

Assembly Name	Version	Cul...	Public Key Token	Proces...
Expression.DevHost	3.0.0.40...		31bf3856ad364e35	x86
Expression.DevHost.resources	3.0.0.40...	en	31bf3856ad364e35	x86
Extensibility	7.0.330...		b03f5f7f11d50a3a	
GAC2008	8.0.0.0		dcc8364b2810e1dc	MSIL
GAC2008	7.0.0.0		dcc8364b2810e1dc	MSIL
GAC2008	5.0.0.0		dcc8364b2810e1dc	MSIL
GAC2008	1.0.0.0		dcc8364b2810e1dc	MSIL
IACore	1.7.622...		31bf3856ad364e35	
IALoader	1.7.622...		31bf3856ad364e35	
IIEExecRemote	2.0.0.0		b03f5f7f11d50a3a	MSIL
IIEHost	2.0.0.0		b03f5f7f11d50a3a	MSIL
IIEHost	2.0.0.0		b03f5f7f11d50a3a	MSIL
Interop.Dso	5.2.0.0		89845dcd8080cc91	MSIL

Figuur 17.1: Global Assembly Cache (iets oudere versie, Win 7)

Microsoft.Web.Administration.resources	7.9.0.0	es	31bf3856ad364e...	MSIL
Microsoft.Web.Administration.resources	7.9.0.0	de	31bf3856ad364e...	MSIL
Microsoft.Web.Administration.resources	7.9.0.0	cs	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation	7.1.0.0		31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	zh-CHT	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	zh-CHS	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	tr	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	ru	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	pt	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	pl	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	ko	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	ja	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	it	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	fr	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	es	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	en	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	de	31bf3856ad364e...	MSIL

Figuur 17.2: Global Assembly Cache (iets oudere versie, Win 8)

- er komen assemblies met identieke namen voor (dit is onmogelijk voor files in gewone directories). Dit maakt het mogelijk om in de GAC aan versie beheer te doen (dit voorkomt de problemen die optreden wanneer incompatibele versies elkaar overschrijven);
- de assemblies hebben een versie nummer. Hierdoor wordt het mogelijk om een programma te koppelen aan een specifieke versie van een assembly;

Niet zichtbaar op de screenshots, maar toch relevant:

- een assembly in de GAC *moet strong named* zijn;
- bij installatie van een nieuwe versie van een assembly kan met een *policy*- file opgegeven worden welke versies vervangen worden: het *redirecten* van assemblies. Deze *policy* files worden ook in de *GAC* geplaatst, en moet dan ook strongly named zijn. Op deze manier kan een upgrade automatisch geactiveerd worden voor programma's die een verouderde assembly gebruiken;
- een programma config file kan dit redirecten overschrijven, en verwijzen naar nog een andere (of de oorspronkelijke) assembly. De config file van het programma heeft voorrang op de assembly policy file.

  Het is belangrijk dat de programma config file voorrang heeft op de assembly policy file: indien een nieuwe versie onverhoopt toch problemen oplevert, kan men per individueel programma beslissen met welke assembly versie men blijft verder werken.
- vanaf Visual Studio 2010 is de GAC meer folder georiënteerd:
  C:\Windows\Microsoft.NET\assembly\GAC_MSIL\GACDLL\v4.0_1.0.0.0__72e3a55536524714

Figuur 17.3: Eigen GAC-dll's, Win 8)

17.3 Ontwikkelen voor en met de GAC

17.3.1 Strong named assemblies

Het MSDN artikel omtrent Strong-named assemblies beschrijft duidelijk *het waarom* omtrent named assemblies:

Strong-Named Assemblies

msdn

A strong name consists of the assembly's identity — its simple text name, version number, and culture information (if provided) — plus a public key and a digital signature. It is generated from an assembly file (the file that contains the assembly manifest, which in turn contains the names and hashes of all the files that make up the assembly), using the corresponding private key. Microsoft® Visual Studio® .NET and other development tools provided in the .NET Framework SDK can assign strong names to an assembly. Assemblies with the same strong name are expected to be identical.

You can ensure that a name is globally unique by signing an assembly with a strong name. In particular, strong names satisfy the following requirements:

- Strong names guarantee name uniqueness by relying on unique key pairs. No one can generate the same assembly name that you can, because an assembly generated with one private key has a different name than an assembly generated with another private key.
- Strong names protect the version lineage of an assembly. A strong name can ensure that no one can produce a subsequent version of your assembly. Users can be sure that a version of the assembly they are loading comes from the same publisher that created the version the application was built with.
- Strong names provide a strong integrity check. Passing the .NET Framework security checks guarantees that the contents of the assembly have not been changed since it was built. Note, however, that strong names in and of themselves do not imply a level of trust like that provided, for example, by a digital signature and supporting certificate.

When you reference a strong-named assembly, you expect to get certain benefits, such as versioning and naming protection. If the strong-named assembly then references an assembly with a simple name, which does not have these benefits, you lose the benefits you would derive from using a strong-named assembly and revert to DLL conflicts. Therefore, strong-named assemblies can only reference other strong-named assemblies.

See Also

Global Assembly Cache | Signing an Assembly with a Strong Name

Figuur 17.4: Strong-named assemblies (MSDN)

Om zelf een strong-named assembly te creeëren gaan we als volgt te werk:

- maak een nieuw project aan (voor dit voorbeeld een class library, hoewel natuurlijk ook alle andere types strong-named kunnen zijn);
- via de project properties in Visual Studio, in de signing farde kiest u:
  - *sign the assembly*;
  - kies een bestaande *strong name key file*, of maak een nieuwe aan. Hierdoor wordt aan het project een *snk*- file toegevoegd. Wanneer u het project herbuild zal de resultaat assembly strong-named zijn.
    - via *sn.exe*- commando kan u ook zelf een key-pair file aanmaken (zie ook How to: Create a Public/Private Key Pair.

- omdat we deze DLL later zullen gebruiken om assembly redirecting te testen zullen we een klasse voorzien die het versie nummer van de assembly teruggeeft.

  System.Reflection.Assembly.GetExecutingAssembly().GetName().Version is een startpunt om de major, minor en build gegevens van een assembly op te vragen.

17.3.2 Gacutil.exe

Dit commando laat toe om de GAC te manipuleren en ondervragen. MSDN beschrijft de parameterlijst, even googlen brengt u op de internetsnelweg.

Indien u een niet *strong-named* dll in de GAC probeert te plaatsen krijgt u het volgende resultaat:

```
Administrator: Visual Studio Command Prompt (2010)

D:\Temp>gacutil -i gacdll.dll
Microsoft (R) .NET Global Assembly Cache Utility.  Version 4.0.30319.1
Copyright (c) Microsoft Corporation.  All rights reserved.

Failure adding assembly to the cache: Attempt to install an assembly without a s
trong name
```

Figuur 17.5: Gacutil -i Strong-named failure

Wanneer de assembly wel strong-named is:

```
D:\Temp>gacutil -i gacdll.dll
Microsoft (R) .NET Global Assembly Cache Utility.  Version 4.0.30319.1
Copyright (c) Microsoft Corporation.  All rights reserved.

Assembly successfully added to the cache

D:\Temp>
```

Figuur 17.6: Gacutil -i Strong-named success

U kan de aanwezigheid in de cache verifëren:

```
Administrator: Visual Studio Command Prompt (2010)

D:\Temp>gacutil -l gacdll
Microsoft (R) .NET Global Assembly Cache Utility.  Version 4.0.30319.1
Copyright (c) Microsoft Corporation.  All rights reserved.

The Global Assembly Cache contains the following assemblies:
  gacdll, Version=1.0.0.0, Culture=neutral, PublicKeyToken=72e3a55536524714, pro
cessorArchitecture=MSIL

Number of items = 1

D:\Temp>
```

Figuur 17.7: Gacutil -l

17.3.3 DLL-versies maken

Een assembly versie bestaat uit major, minor, build en revision getallen. U kan deze instellen via de project properties (application, assembly information):

Figuur 17.8: Assembly versies

U kan de versie van een file opvragen via zijn properties:

Figuur 17.9: File properties

Het is perfect mogelijk om ook deze DLL in de GAC te plaatsen. Op dat moment bevat de GAC twee dll's van dezelfde assembly.

17.3.4 Redirecting assemblies

17.3.4.1 Welke assembly wordt geladen?

Het zoeken naar assemblies wordt beschreven in How the Runtime Locates Assemblies. Hier vindt u onder andere:

The runtime uses the following steps to resolve an assembly reference:

1. Determines the correct assembly version by examining applicable configuration files, including the application configuration file, publisher policy file, and machine configuration file. If the configuration file is located on a remote machine, the runtime must locate and download the application configuration file first.
2. Checks whether the assembly name has been bound to before and, if so, uses the previously loaded assembly.
3. Checks the global assembly cache. If the assembly is found there, the runtime uses this assembly.
4. Probes for the assembly using the following steps:
   a. If configuration and publisher policy do not affect the original reference and if the bind request was created using the Assembly.LoadFrom method, the runtime checks for location hints.
   b. If a codebase is found in the configuration files, the runtime checks only this location. If this probe fails, the runtime determines that the binding request failed and no other probing occurs.
   c. Probes for the assembly using the heuristics described in the probing section. If the assembly is not found after probing, the runtime requests the Windows Installer to provide the assembly. This acts as an install-on-demand feature.

Note There is no version checking for assemblies without strong names, nor does the runtime check in the global assembly cache for assemblies without strong names.

Figuur 17.10: Assembly Loading

Belangrijk hierin op te merken zijn volgende elementen:

- welke versie van toepassing is is afhankelijk van een aantal configuratie of policy files;
- indien geen configuratie of policy redirections gedefinieerd zijn is de at compile time gebruikte versie van toepassing, ook al is er een nieuwere versie in de GAC te vinden;
- indien de assembly al eerder werd geladen, dan wordt deze gebruikt;
- de GAC heeft voorrang op alle andere locaties. Uiteraard enkel van toepassing voor strong-named assemblies, omdat dit een voorwaarde is om in de GAC geplaatst te kunnen worden.

Eventuele uitvoeringsproblemen bij het laden van een assembly kan u documenteren met behulp van fuslogvw.exe.

17.3.4.2 Redirecting met configuration file

Een volledige write-up omtrent assembly redirection vindt u in Redirecting Assembly Versions. Onderstaande tekst is slechts een beperkte weergave van de vermeldde MSDN-entry. Het hier uitgewerkte voorbeeld maakt gebruik van de twee DLL-versies die we in de GAC plaatsten: versie 1.0.0.0 en 2.1.0.0.

- maak een WPF applicatie die refereert naar de recenste versie en gebruik daarin de klasse met het assembly versie nummer om het de assembly versie te tonen;
  - deze DLL wordt trouwens niet naar de *bin* directory gecopieerd;
- voer de toepassing uit en check de assembly versie;
- indien we de toepassing wensen te redirecten naar een andere versie (1.0.0.0) zullen we in de configuratie file wat extra XML voorzien:

The following example shows how to redirect one version of `myAssembly` to another, and turn off publisher policy for `mySecon`

```
<configuration>
   <runtime>
      <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
       <dependentAssembly>
         <assemblyIdentity name="myAssembly"
                           publicKeyToken="32ab4ba45e0a69a1"
                           culture="en-us" />
         <!-- Assembly versions can be redirected in application, publisher policy, or machine configura
         <bindingRedirect oldVersion="1.0.0.0"
                          newVersion="2.0.0.0"/>
       </dependentAssembly>
       <dependentAssembly>
         <assemblyIdentity name="mySecondAssembly"
                           publicKeyToken="32ab4ba45e0a69a1"
                           culture="en-us" />
          <!-- Publisher policy can be set only in the application configuration file. -->
          <publisherPolicy apply="no">
       </dependentAssembly>
      </assemblyBinding>
   </runtime>
</configuration>
```

Figuur 17.11: configredirection

  - het *public key token* kan u op volgende wijzes bekomen:
    * gebruik *gacutil -l*: de opgelijste informatie bevat het token;
    * zoek in de assembly cache folder (als onderdeel van diretory naam);
    * voeg een external tool toe in Visual Studio, zoals beschreven in Visual Studio Tip: Get Public Key Token for a Strong Named Assembly. Het voordeel van deze manier van werken is het copy/paste beschikbaar zijn van het token binnenin Visual Studio.

  - de *culture* tage is optioneel. Indien afwezig (of *neutral*) is de redirect van toepassing op alle cultures (taal);
  - u kan ook ranges opgeven bij het redirecten.

17.3.4.3 Redirecting met policy files

De aangeraden writeup: How to: Create a Publisher Policy Ga op volgende wijze te werk:

- wijzig de config file in de bin-directory van de WPF-applicatie zodat er geen redirection meer toegepast wordt: de gebruikte dll-versie is 2.1.0.0. Check dit door de de gecompileerde versie uit te voeren;
- wijzig de DLL-versie naar 3.0.0.0 en hercompileer de DLL (*niet* de applicatie). De toepassing werkt nog steeds met versie 2.1.0.0.
- maak een *policy.config* file in de directory waar de keypair file staat. Plaats hierin de XML-code om 2.1 te redirecten naar 3.0.
- gebruik de assembly linker *al.exe* zoals beschreven in How to: Create a Publisher Policy om een policy file aan te maken;
  - ikzelf navigeer de commandprompt met *cd* commando's naar de directory waarin zich de *snk* en *policy* file bevinden. Op deze manier vermijd ik lange filenamen (en bijhorende tikfouten) bij het intikken;
  - merkwaardig genoeg zal in de policy-file de *major.minor* van de te vervangen dll worden opgenomen, en *niet* deze van de vervangende dll die nu geïnstalleerd wordt als vervanger;
- plaats de policy file in de GAC. Om hem terug te vinden moet u in de GAC zoeken op de prefix *policy*;

```
Administrator: Visual Studio Command Prompt (2010)

D:\prevdata\PIH\vakken\20102011\programmer solutions\lessen\DemoGAC\GACDLL>al /link:policy
.config /out:policy.2.1.GACDLL.dll /keyfile:gacdll.snk /platform:anycpu
Microsoft (R) Assembly Linker version 10.0.30319.1
Copyright (C) Microsoft Corporation. All rights reserved.


D:\prevdata\PIH\vakken\20102011\programmer solutions\lessen\DemoGAC\GACDLL>gacutil /i poli
cy.2.1.GACDLL.dll
Microsoft (R) .NET Global Assembly Cache Utility.  Version 4.0.30319.1
Copyright (c) Microsoft Corporation.  All rights reserved.

Assembly successfully added to the cache

D:\prevdata\PIH\vakken\20102011\programmer solutions\lessen\DemoGAC\GACDLL>
```

Figuur 17.12: Policy file commando's

- plaats de nieuwe DLL versie in de GAC;

- test uw toepassing: deze moet nu versie 3.0 gebruiken;
- wijzig de config file terug zodat versie 1.0 gebruikt wordt. Dit illustreert de voorrang van de configuration file op de policy file.

17.3.4.4 Redirecting op machine niveau

Dit gebeurt opnieuw met configuration files en maakt geen onderdeel uit van deze cursus.

17.3.4.5 Redirecting redirections

Dit lijkt niet mogelijk te zijn:

- maak een programma dat gebruik maakt van versie 1.0 van een DLL;
- maak een policy file die versie 1.0 vervangt door versie 2.0;
- maak een configuration file die versie 2.0 vervangt door versie 3.0;
- plaats alle versies in de GAC;
- bij uitvoering wordt versie 2.0 gebruikt: herhaald redirecting lijkt dus niet te werken (wat ook overeenstemt met het artikel dat beschrijft hoe een assembly gevonden wordt)

Jef Daels

17.3.5 GAC en project references

Voor een volledige toelichting: How to: Add or Remove References in Visual Studio. Belangrijke is het volgende element:

- GAC assemblies komen *niet* voor in de references dialogen

Dll's gevonden in de GAC worden ook niet opgenomen in de bin-directory van de client-toepassing.

17.3.6 Installeren in de GAC vanuit de Setup (tem VS 2010)

In een Visual Studio installatie project kan u een *special folder* opnemen die verwijst naar de GAC:

Figuur 17.13: GAC en setup

Indien u hier (strong named) assemblies opneemt zullen deze in de GAC van het target toestel geïnstalleerd worden.

17.4 GAC opgave

Implementeer de situatie die hierboven wordt beschreven en hieronder summier herhaald wordt:

- maak een WPF toepassing die gebruik maakt van een eigen DLL. Deze DLL biedt onder andere de mogelijkheid om zijn assembly versie te ondervragen. De applicatie gebruikt dit om de gebruikte DLL versie te tonen;
- alle versies van de DLL komen in de GAC terecht;
- gebruik een config file om te redirecten, u kiest tussen welke versies;
- gebruik een policy file om te redirecten;
- toon aan dat de configuration file redirection voorrang heeft op de policy file redirection.

17.5 Extras

addin om vanuit visual studio assembly redirection te implementeren manifest?

Jef Daels

Hoofdstuk 18

Visual Studio geautomatiseerde broncode manipulaties

18.1 Visual Studio geautomatiseerde broncode manipulaties: duiding

Een moderne IDE helpt de ontwikkelaar bij het produceren van code: zowel het reduceren van tikwerk (de IDE genereert bijvoorbeeld de proxy klasses gebruikt in een WCF-oplossing) als het vermijden van foutieve code (intellisense) zijn hiervan voorbeelden. In dit hoofdstuk komen volgende elementen aan bod: toolbox codeblocks, snippets, macro's, add-ins en (kort) het gebruik van MEF in Visual Studio 2010.

18.2 Relevante namespaces

- EnvDTE: een namespace die informatie omtrent Visual Studio projecten bevat (*alle* types projecten);
  - *Changes and new functionality are contained in the EnvDTE80, EnvDTE90, EnvDTE90a, and EnvDTE100 namespaces.*: er zijn dus upgrades
- VslangProj100: in deze assembly wordt extra informatie omtrent programmeer projecten bijgehouden die men niet kwijt kan in *EnvDTE* (bijvoorbeeld: referenties)
  - *The VslangProj100 namespace defines types used to automate project tasks. These types extend the types defined in the VSLangProj, VSLangProj2, VSLangProj80, and VslangProj90 namespaces.*: ook hier zijn dus verschillende (opeenvolgende) versie (upgrades) beschikbaar.

18.3 Toolbox codeblocks

Wanneer we in het code-window vertoeven is de toolbox meestal leeg. Het is mogelijk om een blok code in de editor te selecteren en deze op de toolbox te slepen, waarna we (via right- click) dit stukje code ook een naam kunnen geven (optioneel). Eens op de toolbox geplaatst kan u dit stukje code overal copiëren door te klikken op zijn toolbox- voorkomen:

Figuur 18.1: Code in de toolbox

18.3.1 Voordeel van toolboxcode

- zeer gemakkelijk op te zetten en te gebruiken: alle manipulaties blijven binnen Visual Studio;

18.3.2 Nadelen van toolboxcode

- de benadering is weinig flexibel: er wordt een letterijke copy aangemaakt die geen enkele ondersteuning biedt bij aanpassingen;
- deze manier van werken is niet vlot porteerbaar tussen toestellen.

18.4 Code snippets

Snippets bieden de mogelijkheid om template code in een source-file te pasten. In de resultaat code is het mogelijk om intelligente vervangingen door te voeren (wijzigen op één plaats van een stukje code zal copy code, elders in het snippet ook wijzigen). Visual Studio wordt aangeleverd met tal van snippets out-of-the box en biedt de mogelijkheid om ook eigen snippets toe te voegen.

18.4.1 Snippets gebruiken

Een snippet wordt geïdentificeerd door een *shortcut* (bijvoorbeeld *propa* voor een attached dependency property). Indien u deze shortcut intikt, gevolgd door twee *TAB* toetsaanslagen wordt de gekoppelde snippet-code in uw bronfile gecopieerd:

```
#region "adres"
prop
```

- DependencyPropertyHelper
- DependencyPropertyKey
- FrameworkPropertyMetadata
- FrameworkPropertyMetadataOptions
- prop
- propa
- propdp
- Properties

Define an attached DependencyProperty
Code snippet for an attached property using DependencyProperty as

Figuur 18.2: propa snippet gebruiken

Bemerk ook het icoontje dat Visual Studio aan een snippet koppelt. Wanneer we met deze snippet code genereren krijgen we het onderstaande resultaat:

```
public static int GetMyProperty(DependencyObject obj)
    {
    return (int)obj.GetValue(MyPropertyProperty);
    }

public static void SetMyProperty(DependencyObject obj, int value)
    {
    obj.SetValue(MyPropertyProperty, value);
    }

// Using a DependencyProperty as the backing store for MyProperty.  This enables animation, styling, bindin
public static readonly DependencyProperty MyPropertyProperty =
    DependencyProperty.RegisterAttached("MyProperty", typeof(int), typeof(ownerclass), new UIPropertyMetada
```

Figuur 18.3: propa snippet resultaat

De tekst met achtergrondkleur zal, indien hij wordt gewijzigd, ook andere tekst aanpassen

(zeker het proberen waard!). Bij het zelf aanmaken van snippets zullen we aangeven hoe deze verschillende codeelementen aan elkaar gekoppeld worden.

18.4.2 Eigen code snippets

Het *propa*-snippet genereert de dependency property registratie en de wrapper methodes. Ik zelf heb bij het definiëren van *attached dependency properties* veelal ook nood aan de propertyChanged callback, waarvan ik de signatuur niet van buiten ken: ik wou dat ik een *snippet* had die deze extra code bevatte. Het is mogelijk om de bestaande snippet aan te passen (mijns insziens minder wenselijk) of een nieuwe snippet aan te maken (een snippet is een gewone ascii-file). We zullen een eigen snippet aanmaken: *propachanged*.

U vindt de geïnstalleerde snippets in subdirectories van *Program Files\Microsoft Visual Studio 10.0\VC#\Snippets\1033*. Het *propa-* snippet wordt gevonden in de *NetFX30* subdirectory. We zullen dit snippet copiëren naar een eigen directory waarna we het snippet zullen aanpassen:

- copieer *propa.snippet* naar een eigen snippet folder;
- pas de beschikbare snippet informatie aan;
- pas de code aan en voorzie een change routine die gekoppeld wordt via de metadata, zodat de property-naam onderdeel uitmaakt van de change-routine, en mee gewijzigd wordt indien u de property naam wijzigt.

Opdat een snippet in Visual Studio gekend zou zijn moet u deze bibliotheek importeren:

- *Tools.Code Snippets Manager*;
- kies de taal waarvoor uw snippet werd ontwikkeld, en voeg uw snippet directory toe.

Hierna kan u uw zopas aangemaakt snippet testen.

18.4.3 Voordelen van snippets

- deze manier van werken kan gedeeld worden door verschillende ontwikkelaars (u vind op het internet vele voorbeelden);
- de snippet-code ondersteunt een vorm van pseudo parametristatie (en is dus iets dynamischer in gebruik dan de toolbox code);

18.4.4 Nadeel van snippets

- echte parametrisatie is nog niet mogelijk: het is bijvoorbeeld onmogelijk om een snippet te maken die in zijn textexpansie het huidige tijdstip opneemt;

Indien u de cursustekst niet leuk vond kan u ook op Extending Visual Studio Part 1 - Creating Code Snippets terecht.

Jef Daels

18.5 Macro's in Visual Studio

Net zoals Office (Visual Basic for Applications) biedt Visual Studio de mogelijkheid om gebruikershandelingen te registereren zodat ze op een later tijdstip opnieuw uitgevoerd kunnen worden: *macro*'s. Omdat deze commando's als programma code worden bewaard kunnen ze ook aangepast worden om ook in niet identieke situaties bruikbaar te zijn. Ikzelf gebruik macro's vooral om voorbeeld code te genereren die ik dan (aangepast) overneem bij het ontwikkelen van Add-ins. Op die manier kan het onderzoeken van het Visual Studio objectmodel sterk beperkt worden (net zoals in Office worden macro's geregistreerd met Visual Basic).

18.5.1 Macro samples

Vanonder *Tools.Macros.Macro Explorer* krijgen we toegang tot de beschikbare macro's. De samples bevatten (bruikbare) voorbeeld macro's die richtinggevend zijn voor de mogelijkheden. U kan ze uitvoeren (double click) of editeren (via right click). Belangrijk hierbij is dat ze ook debugbaar zijn: u kan breakpoints plaatsen om de logica van de macro na te volgen:

- onder *DevStudio6Editor* vindt u de macro *CommentOut*. Deze zal een highlighted stuk code in commentaar plaatsen. Probeer dit in uw programma, normaal gezien zal dit niet werken. Gebruik de debugger mogelijkheden om na te pluizen waarom dit niet werkt, en pas dit aan.

18.5.2 Eigen macro's

In deze les zullen we zelf geen eigen macro's ontwikkelen maar onmiddellijk overstappen naar het ontwikkelen van addIn's.

18.5.3 Voordelen van macro's

- programmeer kennis is niet nodig om een macro op te nemen (*Tools.Macros* helpt u verder);

- een macro kan een zeer gemakkelijke manier zijn om startcode te bekomen voor het maken van een Add-In (we moeten het objectmodel van Visual Studio niet zelf uitpluizen).

18.5.4 Nadeel van macro's

- het macro recorden beperkt zich tot het registreren door middel van VB-code (voor een nMCT-student niet echt onoverkomelijk);

- macro's bieden weinig mogelijkheden tot visuele feedback. Dit is voor ons de reden om geen eigen Macro te ontwikkelen en onmiddellijk over te stappen naar Add-ins.

18.6 Visual Studio Addin's

Een Addin nestelt zich in de toolbar of het menu van uw Visual Studio omgeving en manipuleert uw Visual Studio solution. Omdat het foutief manipuleren van Visual Studio grote problemen kan veroorzaken (Visual Studio onbruikbaar kan maken) wordt in een test-fase gewerkt met een Experimental Build. Indien er iets fout loopt gebeurt dit in deze copy omgeving, en blijft uw originele Visual Studio bruikbaar.

- om addins te ontwikkelen installeert u Microsoft Visual Studio SDK;
- u kan de experimental hive (de registry entries gebruikt door de experimental build) resetten door *Visual Studio 2010 SDK.tools.Reset the Microsoft Visual Studio Experimental Hive* te kiezen.

18.6.1 Creatie van een Visual Studio Add-in

Om een Addin aan te maken gaan we als volgt te werk:

- maak een nieuw project aan met als projecttype *Other Project Types.Extensibility.Visual Studio Add-in*;
  - bij het doorlopen van de wizard kiezen we als programmeertaal *VB.NET*. Bij het programmeren van de Add-In's worden veel properties als *Object* opgeleverd. In die situaties is het handig om gebruik te maken van de late binding mogelijkheden van VB.NET;
    - u kan in *C#*een gelijkaardig effect als *late binding* bekomen door gebruik te maken van het keyword dynamic. Ikzelf vind het gemakkelijker om gewoon in VB.NET te programmeren:
      - automatische late binding;
      - de macro gegenereerde code is VBA-code, en daardoor meer lijkend op VB.NET dan op *C#*.
  - application host is de toepassing waarbinnen de add-in bruikbaar moet zijn;
  - bij het invullen van 'Choose Add-in Options' is de commentaar bij elke checkbox volgens mij duidelijk genoeg om zelfstandig de keuzes te maken;

18.6.2 Testuivoering van een Add-in

Indien u het project nu opstart wordt een nieuwe Visual Studio gestart waarin deze Addin actief is. Fouten (door de nieuwe Add-in) in deze omgeving hebben geen invloed op de originele Visual Studio omgeving.

Indien de add-in niet geladen wordt in de extra gestartte omgeving kan het eventueel helpen om in de startup options van het project volgende commandline parameter mee te geven:

- */ranu /rootsuffix Exp*

De aangemaakte Add-In bevat standaardaard al flink wat code. We zullen ons concentreren op de *Connect* file:

- de constructor van de Connect- klasse laat toe om extra initialisatie code uit te voeren;
- *OnConnectionConnect!OnConnection* wordt uitgevoerd wanneer de Add-In gekoppeld wordt aan Visual Studio. Normaal gezien zullen we deze code ongewijzigd laten;
- de methode *Exec* wordt uitgevoerd wanneer de Visual Studio gebruiker de Add-In start, en bevat de eigenlijke logica van de Add-In.

18.6.3 Een lijnenteller Add-In

We zullen de add-in aanpassen zodat we in een listview per project in de solution alle files en hun aantal lijnen tonen. Met deze Add-In krijgen we een idee over de grootte van een solution.

- maak een formulier dat deze informatie zal tonen en start dit formulier vanuit de *Exec* methode (van de Connect- klasse van de Add-In);
  - momenteel is het enkel mogelijk om *WinForms* Add-in formulieren aan te maken;
  - in de ctor geeft u wellicht best het applicationobject van de addin mee (een object dat de applicatie, in dit geval Visual Studio bevat);
- *EnvDTE* is de namespace waarin u de hier nodige klasses (intefaces) terugvindt. In combinatie met de intellisense en wat zoeken kan u de lijnenteller hopelijk opzetten;
  - projectitems kunnen genest zijn: zo zijn designer generated code files projectitems van de origineel toegevoegde files;
  - niet elk projectitem heeft een overeenkomstige file;

Wees kritisch omtrent uw output: indien er maar één file gevonden wordt in uw project is er zeer veel kans dat uw Add-In foutief werkt!

18.6.4 Reference historiek Add-In

EnvDTE bevat enkel die informatie die gemeenschappelijk is aan alle Visual Studio projecten (setup, VB.NET, website, silverlight, phone, ..). Extra programmeer specifieke informatie vindt men in *VsLangProj*. Instanties van *EnvDTE* types kunnen gecast worden naar hun *VsLangProj* tegenhanger via de *.object* property van het *EnvDTE* object.

Als oefening zullen we een Add-in maken die voor elk project in een solution een file aanmaakt (datum staat in de filenaam) waarin de gebruikte referenties worden weggeschreven. Normaal gezien zou ik dit in een database registereren, maar op deze manier zijn we ook verplicht een extra file aan een project toe te voegen (een goede oefening). Met dergelijke informatie is het mogelijk te achterhalen welke projecten potentieel probleem hebben bij het verlopen of upgraden van een dll.

- maak terug een Visual Basic .NET Add-In;
- maak een macro voor de volgende acties:
  - selecteer een project in een solution;
  - voeg aan dit project een nieuwe file toe;
  - tik iets in deze nieuwe file
- in de Add-In ontwikkelen we code die per project een reference historiek file aanmaakt en daarin de references wegschrijft. Gebruik de macrocode als startcode om dit te realiseren.
  - de macro code werkt als macro code. In een Add-In werkt de code, om een nieuwe file toe te voegen niet. Gelukkig wordt in ItemOperations.AddNewItem Method duidelijk beschreven hoe de argumenten voor het toevoegen moeten geschreven worden;
  - uw code moet gebruik maken van een solution name en project name (enige nazicht in de macro code maakt duidelijk waarom). Dit komt niet automatisch, u zal hier wat code moeten voor schrijven;
- indien u de nieuw aangemaakt file wenst te saven kan u in de macro best ook een *save all* instructie opnemen.

Jef Daels

18.7 Visual Studio events

Gedurende een Visual Studio sessie doen zich vele events voor waar een Add-In kan op inpikken: het toevoegen van items, het openen en of sluiten van projecten zijn hiervan voorbeelden. Indien u hierop wenst in te pikken kan u dit op volgende wijze:

```
Private solEvents As SolutionEvents
Private tekstEvents As TextEditorEvents
Private Sub startupComplete()
    solEvents = _applicationObject.Events.SolutionEvents
    AddHandler solEvents.ProjectAdded, AddressOf projAdded
    AddHandler solEvents.Opened, AddressOf solOpened

    tekstEvents = _applicationObject.Events.TextEditorEvents
    AddHandler tekstEvents.LineChanged, AddressOf LineChanged
End Sub
Private Sub solOpened()
    MessageBox.Show("sol opened")
End Sub
Private Sub projAdded(ByVal proj As EnvDTE.Project)
    MessageBox.Show("proj added")

End Sub
Private Sub LineChanged(ByVal start As EnvDTE.TextPoint, ByVal eind As Env
    MessageBox.Show("item added")
End Sub
```

Figuur 18.4: Eventhandling in een Add-In

Belangrijk in bovenstaande code zijn volgende elementen:

- de events- variabelenw waarvoor events worden opgevangen moeten op moduleniveau geplaatst worden (zie ook Getting DocumentEvents after solution is opened).
- het koppelen van events gebeurt in VB.NET niet door middel van de +=- operator maar met behulp van een *AddHandler*- instructie. Uiteraard moet ook hier de signatuur van de gekoppelde routine geschikt zijn;
- het grootste probleem is wellicht het vinden van het gepaste event block. In het voorbeeld komen volgende zaken aan bod:
  - het openen van een solution;
  - het toevoegen van een project
  - het wijzigen van een lijn code

18.8 Deployment van een Add-In

Indien een Add-In naar behoren werkt wensen we deze niet langer in de experimental hive uit te voeren: we wensen de Add-In te gebruiken in een normale Visual Studio omgeving, of de add-In te verspreiden naar andere computers. Om dit te realiseren gaan we als volgt te werk:

- lees How to create and load add-ins in Visual Studio .NET by using Visual $C^{\#}$.NET or in Visual Studio 2005 by using Visual $C^{\#}$2005 en voer uit wat daarin beschreven staat (er ontbreken belangrijke stappen, maar een goed begrijper met voorkennis kan die wel oplossen)
  - pas deze manier van werken toe voor de references Add-In;
  - bij mij werkt het niet;
- de ouderwetse manier (zonder .msi file) die enkel werkt indien u Visual Studio installeerde (omdat we de referenties niet meeinstalleren):
  - via *Tools.Options* achterhalen we de directories waarin Add-Ins gezocht worden:

Figuur 18.5: Addin directories

  - in mijn installatie bestond de *Addins*- directory niet: ik heb deze manueel aangemaakt

  - copieer de *.addin* file die Visual Studio in uw bronfile (bij de code files) plaatste, tesamen met de *.dll* uit de compilatie directory naar de Add-In directory die u koos (indien u niet één van de aanwezige directories koos moet u uw eigen directory toevoegen)
  - bij het heropstarten zal visual studio de AddIn terugvinden en integereren;

- u kan het al dan niet activeren van Add-Ins aansturen vanonder *Tools.Add-in Manager*.

18.9 Visual Studio en MEF: extentions

Zoals in het hoofdstuk MEF werd aangehaald is Visual Studio 2010 MEF-gebaseerd. De technologie op zich is relatief gemakkelijk te begrijpen, maar het voorzien van extenties in een complexe software als Visual Studio is toch niet evident. Extending the Editor is een potentieel startpunt om u hierin te verdiepen. In het vervolg van deze tekst zullen we de intellisense uitbreiden om een gemakkelijk commentaarregels aan code toe te voegen. Uiteindelijk wordt ook toegelicht hoe deze extenties kunnen *gedeployed* worden.

Hoewel we nu enkel zullen ingrijpen op de intellisense kan er natuurlijk veel meer gerealiseerd worden. Een aantal voorbeelden (met broncode) hiervan kan u terugvinden op MSDN Developer Samples: Learn Through Code waarbij u met behulp van het zoekvenster zoekt op *VSX*

18.9.1 cmnt extention

Visual Studio gebruikt intellisense om de ontwikkelaar bij het coderen mogelijke aanvullingen te presenteren. Dit verhoogt de tiksnelheid, vermijdt fouten en vereist gememoriseerde namen van de ontwikkelaar. Het is de bedoeling om via intellisense een commentaarregel te kunnen toevoegen waarin zowel de naam van de ontwikkelaar als het tijdstip van toevoegen voorkomt:

Figuur 18.6: Cmt intellisense

- dit kan niet met snippets gerealiseerd worden omdat deze uit vaste tekst bestaan: de auteur en het tijdstip zouden vaste waardes zijn die achteraf gewijzigd moeten worden.

18.9.2 cmt extention implementatie

- Walkthrough: Displaying Statement Completion diende als leidraad om mijn eigen *cmnt* completion aan te maken: volg de volledige beschrijving;

- het voorbeeld past intellisense toe op *PlainText* files en werkt dus enkel op txt-files, en niet in $C^{\#}$of VB- code files. Om dit te verhelpen moet u het *ContentType* aanpassen (op twee! plaatsen);
  - natuurlijk moet u bij het overnemen van de code de gepaste wijzigingen aanbrengen: u moet de *completions* van het voorbeeld aanpassen. Wie de code begrijpt zal daar weinig problemen bij ondervinden;
  - u zal moeten opzoeken wel ContentType u dan wel dient te gebruiken.
- ik heb niet de pretentie die code voldoende te begrijpen om ze zelf van scratch te ontwikkelen.

18.9.3 cmt deployment

Indien u een extention wenst te installeren is copy deployment onvoldoende. Er is nood aan een *VSIX*- file, die bij dubbelklikken de extentie installeert. Het aanmaken van een VSIX-file wordt beschreven in Packaging an Extension in an Empty VSIX Project. Beter dan die telst kan ik de topic niet beschrijven.

18.9.4 Extentions en Visual Studio

Visual Studio heeft een *Tools.Extention Manager*. Hiermee kunnen geïnstalleerde extentions gedisabled of uninstalled worden.

18.10 Relevante literatuur

- Creating and Using IntelliSense Code Snippets: alles wat u ooit over snippets wou weten, en nog veel meer!
- Getting Started with Extending Visual Studio
- Extending the Editor
- VSX Home on Code Gallery

Jef Daels

18.11 Codemanipulatie opgaves

- ontwikkel een snippet waarmee een attached dependency property wordt aangemaakt, aangevuld met een change- handler. De change handler routine naam is gekoppeld aan de property naam (en wordt dus gewijzigd indien de property naam wordt gewijzigd);
- ontwikkel een Add-In om voor elke file in elk project van een solution het aantal codelijnen te tellen;
- ontwikkel een Add-In om voor elk project in een solution de referenties in een file bij te houden. Elke uitvoering maakt een eigen versie van deze file. U neemt de datum en het tijdstiop hiervoor op in de filenaam;
- (optioneel) maak een extention die toelaat om een commentaar lijn (zoals hierboven beschreven) toe te voegen aan een code window.

Jef Daels

Figuren

2.1 Extention method definition . . . 13
2.2 Extention method gebruik . . . 13
2.3 Extention method gebruik: fadeto . . . 14
2.4 Await demo . . . 15
2.5 Await demo async methode . . . 15

3.1 De klasse nMCTDocent . . . 17
3.2 DisplayMemberPath binding . . . 18
3.3 BindingPathError . . . 18
3.4 ILSpy DisplayMemberPath info . . . 19
3.5 ViewModel klasse . . . 20
3.6 Application Resources . . . 20
3.7 Window binding . . . 20
3.8 DataContext binding . . . 21
3.9 ItemsSource declaratie . . . 21
3.10 Observable hoofding . . . 22
3.11 Collection hoofding . . . 22
3.12 ItemsSource binding . . . 22
3.13 ItemTemplate instellen . . . 22
3.14 nMCTDocent datatemplate . . . 23
3.15 nMCTDocenten in een ComboBox . . . 23
3.16 RelativeSource binding . . . 24

4.1 Delegate definitie . . . 26
4.2 Delegate demo: uppercase en lowercase . . . 26
4.3 Event definitie (zonder guidelines) . . . 28
4.4 Event abonnering . . . 28
4.5 Event afvuren (zonder guidelines) . . . 29
4.6 Static event definitie (volgens de guidelines) . . . 31
4.7 Static event client . . . 31
4.8 Asynchrone verwerking demo . . . 34
4.9 AsyncCallback demo . . . 35
4.10 UI-thread problem . . . 35
4.11 Dispatcher demo . . . 36
4.12 Await- keyword . . . 37

4.13 XMLViewer en code plumbing . . . 40
4.14 Anonymous method voorbeeld . . . 42
4.15 Lambda expressie voorbeeld . . . 42

5.1 ComponentModel attributen ($C^{\#}$) . . . 46
5.2 Assembly reflection ($C^{\#}$) . . . 47
5.3 Strongly typed instantiatie . . . 48
5.4 Activator.CreateInstance ($C^{\#}$) . . . 48
5.5 Activator.InvokeMember ($C^{\#}$) . . . 49

6.1 Commandline compilatie: $C^{\#}$broncode . . . 54
6.2 $C^{\#}$commandline compilatie en uitvoering . . . 55
6.3 Commandline compilatie: VB.NET broncode . . . 55
6.4 VB.NET commandline compilatie voorbeeld . . . 56
6.5 $C^{\#}$System.CodeDom.Compiler namespace . . . 57
6.6 CodeDOM . . . 58
6.7 ILDasm output . . . 59
6.8 ILSpy output . . . 60
6.9 Reflector output . . . 61
6.10 Obfuscated output via Reflector . . . 62
6.11 Tekenen van een functie . . . 63

7.1 NQueens oplossen . . . 66
7.2 Sudoku spelregels: een opgave . . . 67
7.3 Sudoku spelregels: een oplossing . . . 68
7.4 Sudoku routed events demo . . . 70
7.5 Awaiting een oproep . . . 71
7.6 TaskCompletionSource: SetResult . . . 71
7.7 TaskCompletionSource demo . . . 72
7.8 4 op een rij: bordwaarde . . . 73
7.9 Minimax algoritme . . . 75

8.1 Attached dependency property in XAML . . . 80
8.2 Attached dependency property definitie ($C^{\#}$) . . . 81
8.3 F1 infrastructuur ($C^{\#}$) . . . 82
8.4 CLR binding error . . . 84
8.5 Dependency property syntax 1 ($C^{\#}$) . . . 84
8.6 Dependency property syntax 2 ($C^{\#}$) . . . 85
8.7 Breedte animatie ($C^{\#}$) . . . 85
8.8 Background animatie . . . 86
8.9 Transformatie van een Button ($C^{\#}$) . . . 87
8.10 Transformatie animatie ($C^{\#}$) . . . 88
8.11 ValueConverter definitie ($C^{\#}$) . . . 89
8.12 ValueConverter XAML ($C^{\#}$) . . . 90
8.13 XAML binding dependency property . . . 92

Jef Daels

9.1 Window met eigen caption . . . 94
9.2 Window met WindowStyle none . . . 95
9.3 Checkbox states visualisatie . . . 98
9.4 Checkbox states via Blend . . . 98
9.5 State animaties in XAML . . . 99
9.6 State en Parte opgave: verkeerslicht . . . 99

10.1 Height en applicationsettings . . . 105
10.2 Enumeration definitie . . . 107

11.1 Sudoku in PDF . . . 110

12.1 Kwadraat code . . . 114
12.2 Kwadraten selecteren . . . 115

14.1 AOP: parameter annotatie . . . 123
14.2 PostSharp en ILSpy . . . 125
14.3 PDFStamper performance counter . . . 127
14.4 Performance counters in Visual Studio . . . 128

17.1 Global Assembly Cache (iets oudere versie, Win 7) . . . 136
17.2 Global Assembly Cache (iets oudere versie, Win 8) . . . 136
17.3 Eigen GAC-dll's, Win 8) . . . 137
17.4 Strong-named assemblies (MSDN) . . . 138
17.5 Gacutil -i Strong-named failure . . . 139
17.6 Gacutil -i Strong-named success . . . 139
17.7 Gacutil -l . . . 139
17.8 Assembly versies . . . 140
17.9 File properties . . . 140
17.10 Assembly Loading . . . 141
17.11 configredirection . . . 142
17.12 Policy file commando's . . . 143
17.13 GAC en setup . . . 145

18.1 Code in de toolbox . . . 148
18.2 propa snippet gebruiken . . . 149
18.3 propa snippet resultaat . . . 149
18.4 Eventhandling in een Add-In . . . 155
18.5 Addin directories . . . 156
18.6 Cmt intellisense . . . 157

Index

Action, 41
Activator, 48
    CreateInstance, 48
Add-ins, 152
AddHandler, 96, 155
addOwner, 83
AddressOf, 29
AffectsMeasure, 91
AffectsRender, 91
al.exe, 143
Alfa Beta pruning, 75
Animaties, 85
Animation
    ColorAnimation, 86
    DoubleAnimation, 85
    Transformation, 87
Anonymous method, 42
AOP, 123
    ContextBoundObject, 123
Application settings, 105
Assembly, 45
Async, 37
async, 15
AsyncCallback, 34
Asynchrone verwerking, 34
    UI-thread, 36
Attach to proces, 116
Attached dependency property, 79
Attribute
    Category, 46
    DefaultEvent, 46
    DefaultProperty, 46
    DefaultValue, 46
    Description, 46
Await, 37
await, 15
backtracking, 65
BeginInvoke, 34
Behavior, 103
    Drag, 102
    Grip, 103, 105
Behaviors, 101
BesteZet, 74
Blend, 88
Bootstrapper Project, 121
BordWaarde, 73
Brush, 86
bubbling, 32

CanConvertFrom, 90
Category, 46
ccCaption, 95
Closed, 106
cmnt token, 157
Code snippet, 149
CodeDomProvider, 57
Color, 86
ColorAnimation, 86
CompilerParameters, 57
CompilerResults, 58
Connect, 153
    Connect, 153
    Exec, 153
ContextBoundObject, 123
contravariance, 27
Convert, 89
ConvertBack, 89
ConvertFrom, 90
covariance, 27
CreateInstance, 48
CreateProvider, 57

DataContext, 19
DefaultEvent, 46

Jef Dacis

DefaultProperty, 46
DefaultValue, 46
Delegate, 25, 45
    Asynchrone verwerking, 34
    BeginInvoke, 34
    codeplumbing, 39
    events, 27
    multithreading, 33
Dependency properties, 79
DependencyProperty, 81
    addOwner, 83
    RegisterAttached, 81
Description, 46
deserialisatie, 89
Dispatcher, 36
DisplayMember, 18
DisplayMemberPath, 18
DoubleAnimation, 85

EAP, 37
EnvDTE, 147, 153
Event based Asynchronous Pattern, 37
EventArgs, 30
Events
    +=, 28
    AddHandler, 29
    AddressOf, 29
    event keyword, 28
    EventArgs, 30
    guidelines, 30
    handles, 28
    onEvent, 30
    preview, 32
    RaiseEvent, 29
    routed events, 32
    static events, 31
    tunneling event, 32
Events:bubbling event, 32

F1URL, 80
Func, 41

GAC, 135
Gacutil.exe, 139
Generics, 30
GetCustomAttributes, 49
GetLanguageFromExtension, 57
GetTemplateChild, 96
Ghost data, 33
Global Assembly Cache, 135
Grip, 103

IAOPValidatie, 125
IAsyncResult, 34
IL, 58, 59
ILDasm, 59
inconsistent read, 33
InstallShield, 119
instructies
    lock, 33
intermediate language, 59
InvokeMember, 49
ItemsControl, 21, 22
ItemsSource, 21
ItemsTemplate, 22
iTextSharp, 109
IValueConverter, 89, 90
    Convert, 89
    ConvertBack, 89

Lambda expression, 42
LinearGradientBrush, 89
LINQ, 42
Lost update, 33

Macros, 151
Main, 54
MEF, 157
MenuItem, 52
MiniMax, 73
    BesteZet, 74
    BordWaarde, 73
    MiniMax, 75
    Pruning, 75
MouseDownEvent, 102
MouseDragElementBehavior, 102
MouseMoveEvent, 102
MSI, 119
Multithreading, 33

N-Queen, 66

Namespace
System, 25
System.CodeDom.Compiler, 53
System.ComponentModel, 45, 46, 79
System.Reflection, 45
System.Windows, 79
System.Windows.Data, 79
System.Windows.Media, 79
nMCTDocent, 17

Obfuscation, 62
ObservableCollection, 17
OnApplyTemplate, 94, 96
OnEntry, 124, 125
OnException, 125
OnMethodBoundaryAspect, 124
OnRender, 91
OpenFileDialog, 47

PARTS, 94
patterns
strategy pattern, 39
Performance counter, 127
PerformanceCounterType, 128
plugin, 45
policy, 137
PostSharp, 123, 124
OnEntry, 124
OnException, 125
OnMethodBoundaryAspect, 124
preview, 32
PreviewKeyDownEvent, 69
Process, 82
propa, 149
propachanged, 150
PropertyPath, 86
Pruning, 75

Query Notification, 131

redirect, 137
Reflection, 45
Reflector, 61
registry, 135
Resize, 104
runtime compilatie, 53
RuntimeInitialize, 126

Server Explorer, 128
signatuur, 27
sn.exe, 138
Snippet, 149
propa, 149
snippet
propa, 81
propdp, 81
snk, 138
SQL Server, 113
SQL Server Datatools, 113
States, 97
StoryBoard, 86
TargetNameProperty, 86
TargetPropertyProperty, 86
String, 54
Strong name, 138
Sudoku, 110
System, 25, 54
System.CodeDom.Compiler, 53, 57
System.ComponentModel, 45, 46, 79
System.Reflection, 45
System.Windows, 79
System.Windows.Data, 79
System.Windows.Media, 79

Table Value Functions, 115
Task, 71
TemplatePart, 94
Themes, 96
Thread
Sleep, 38
Toolbox code, 148
Transformation, 87
TryParse, 90
tunneling, 32
Type.GetCustomAttributes, 49
Type.InvokeMember, 49
TypeConverter, 89, 90

UI-thread, 36
using, 54

ValueConversion, 89

Jef Dach

Version, 139
Visual Studio, 128
VisualStateManager, 99
VisualTreeHelper, 96, 103
    GetParent, 96
VSIX, 158
VslangProj100, 147

Window, 50
WindowStyle, 95
WiX, 119

XMLViewer, 39

Programmer solutions

Jef Daels

Jef Daels

Inhoud

1 Cursus aanpassingen **11**

2 Random topics **13**
2.1 Extention methods . . . 13
2.1.1 Gebruik van Extention methods . . . 14
2.1.1.1 LINQ en extention methods . . . 14
2.1.1.2 Animaties en extention methods . . . 14
2.1.2 Extention method surprises . . . 14
2.2 Partial classes . . . 14
2.3 Partial methods . . . 14
2.4 Transformations . . . 14
2.5 Awaitable methodes . . . 15
2.6 (XML) serialisatie in .NET . . . 15
2.7 Var keyword . . . 16
2.8 TryParse methodes . . . 16
2.9 Dynamic keyword . . . 16
2.9.1 lock en volatile keywords . . . 16
2.9.2 WPF toolkit . . . 16
2.9.3 Application settings in WPF . . . 16
2.9.4 conditional[debug] . . . 16
2.9.5 custom build . . . 16
2.9.6 sys tools . . . 16

3 Op aanvraag: Data Binding in WPF **17**
3.1 De klasse nMCTDocent . . . 17
3.2 DisplayMemberPath: the old way . . . 18
3.3 DataContext: default binding source . . . 19
3.3.1 DataContext: duiding . . . 19
3.3.2 Datacontext: instellen op een window . . . 20
3.3.2.1 ViewModel voor de demo . . . 20
3.3.2.2 Instellen van de Window DataContext . . . 20
3.3.2.3 Binden van de ItemsControl.ItemsSource Property . . . 21
3.3.2.4 Instellen van de ItemsControl.ItemTemplate Property . . . 22
3.4 Niet DataContext binding sources . . . 23
3.5 TypeConverter . . . 24

4 Delegates **25**
4.1 Delegates: duiding . . . 25
4.1.1 Relevante namespace . . . 25
4.2 Delegates: technische intro . . . 26
4.2.1 Demo-code en toelichting . . . 26
4.2.2 Covariance en contravariance . . . 27
4.2.2.1 Covariance . . . 27
4.2.2.2 Contravariance . . . 27
4.3 Delegates en events . . . 27
4.3.1 Definitie van een event . . . 28
4.3.2 Abonneren (to subscribe) op een event . . . 28
4.3.2.1 Event subscription: verschil met VB.NET . . . 29
4.3.3 Afvuren van een event . . . 29
4.3.3.1 Afvuren van een event: verschil met VB.NET . . . 29
4.3.4 Meerdere clients voor hetzelfde event . . . 30
4.4 Event Guidelines! . . . 30
4.5 Omtrent Events: . . . 31
4.5.1 Static events . . . 31
4.5.2 Delegates of Interfaces: .NET of JAVA events . . . 32
4.5.3 Routed events . . . 32
4.6 Delegates en multithreading . . . 33
4.6.1 Multithreading: wat bedoelt u? . . . 33
4.6.2 Multithreading: iets voor ons? . . . 33
4.6.3 Asynchrone processing en delegates . . . 34
4.6.3.1 Asynchroon, geen AsyncCallback . . . 34
4.6.3.2 Asynchroon, met AsyncCallback, zonder UI . . . 35
4.6.3.3 Asynchroon, met AsyncCallback, met UI . . . 36
4.6.4 Multithreading met de Thread klasse . . . 36
4.6.5 Event based Ansynchronous Pattern (EAP) . . . 37
4.6.6 Asynchroon programmeren made easy . . . 37
4.6.7 CancellationToken . . . 38
4.6.8 Thread.Sleep . . . 38
4.6.9 Meer info? . . . 38
4.7 Code plumbing, aka 'the strategy pattern' . . . 39
4.7.1 Codeplumbing: een XMLViewer . . . 39
4.7.2 Codeplumbing pitfalls . . . 40
4.8 Delegates: faits divers . . . 41
4.8.1 Action en Func . . . 41
4.8.1.1 Action . . . 41
4.8.1.2 Func . . . 41
4.8.2 Meer dan (begin)-invoke . . . 41
4.8.3 Timers . . . 41
4.8.4 Asynchrone eventverwerking . . . 42
4.8.5 Anonymous methods (.NET 2.0) . . . 42

4.8.6 Lambda expressions (.NET 3.0) . . . 42
4.9 Delegates: niet voor elk programma een geschikte oplossing . . . 43
4.10 Delegates: opgaves . . . 43

5 Reflection **45**
5.1 Reflection: duiding . . . 45
5.1.1 Relevante namespaces . . . 45
5.2 Reflection: attributen voorbeeld . . . 46
5.3 Reflection in .NET: meer dan attributen . . . 47
5.3.1 Reflection in .NET: start . . . 47
5.3.2 Hoe een instantie maken van een klasse in een assembly? . . . 48
5.3.3 .NET en attributes . . . 49
5.3.3.1 .NET en attribute informatie opvragen . . . 49
5.3.3.2 Hoe zelf Attribute klasses maken in .NET? . . . 49
5.4 Een eigen plugin- infrastructuur! . . . 50
5.4.1 Een eigen plugin- infrastructuur: vereisten . . . 50
5.4.2 Een eigen plugin- infrastructuur: mogelijke oplossing . . . 50
5.4.3 Een eigen plugin- infrastructuur: implementatie . . . 51
5.4.4 Eigen plugin infrastructuur: demo . . . 52
5.5 Namespaces met interessante attributen . . . 52
5.6 Reflection: opgaves . . . 52

6 Runtime compilatie **53**
6.1 Runtime compilatie: duiding . . . 53
6.1.1 Relevante namespaces . . . 53
6.2 .NET compilaties starten . . . 54
6.2.1 Command line compilatie . . . 54
6.2.1.1 Command line compilatie: *C#* . . . 54
6.2.1.2 Command line compilatie: VB.NET . . . 55
6.2.1.3 Een extraatje omtrent MSBuild . . . 56
6.2.2 Runtime compilatie: System.CodeDom.Compiler . . . 57
6.3 CodeDom . . . 58
6.4 System.Reflection.emit . . . 58
6.5 Het Roslyn project . . . 58
6.6 Assembly code inspecteren . . . 59
6.6.1 ILDasm . . . 59
6.6.2 ILSpy . . . 60
6.6.3 Reflector . . . 61
6.6.4 Obfuscatie . . . 62
6.6.4.1 Confuser . . . 62
6.7 Opgaves . . . 63
6.7.1 Een tekenprogramma . . . 63
6.7.2 ILSpy . . . 63
6.7.3 Obfuscatie . . . 63

7 Backtracking **65**
7.1 Backtracking: duiding . . . 65
7.1.1 Relevante namespaces . . . 65
7.1.2 Backtracking: principe . . . 65
7.2 N-Queens probleem . . . 66
7.2.1 Probleemstelling, synchrone oplossing . . . 66
7.2.2 Asynchroon: BeginInvoke/4 . . . 66
7.2.3 Asynchroon: async/await . . . 67
7.3 Sudoku's oplossen . . . 67
7.3.1 Sudoku: spelregels . . . 67
7.3.2 Software vereisten . . . 68
7.3.3 Sudoku oplos algoritme . . . 68
7.3.3.1 Constraint satisfaction solver . . . 69
7.4 Sudoku routed events introductie . . . 69
7.5 Async methodes: TaskCompletionSource . . . 70
7.5.1 Task- klasses . . . 71
7.5.2 TaskCompletionSource- klasses . . . 71
7.6 Het MiniMax algoritme (geen examen stof) . . . 73
7.6.1 MiniMax benadering voor vier op een rij . . . 73
7.6.1.1 BordWaarde . . . 73
7.6.1.2 BesteZet . . . 74
7.6.1.3 MiniMax methode . . . 75
7.6.2 Minimax optimalisaties . . . 75
7.6.2.1 Alfa-Beta pruning . . . 75
7.6.2.2 Multi- threading . . . 76
7.7 Backtracking opgaves . . . 77

8 Dependency properties **79**
8.1 Dependency properties: duiding . . . 79
8.1.1 Relevante namespaces . . . 79
8.2 F1URL: een help infrastructuur . . . 80
8.2.1 F1URL gebruik in XAML . . . 80
8.2.2 F1URL definitie . . . 81
8.2.3 F1URL infrastructuur . . . 82
8.3 Dependency properties: technisch . . . 83
8.4 Dependency property XAML binding . . . 84
8.5 Dependency property syntax . . . 84
8.6 Dependency properties en animaties . . . 85
8.6.1 DoubleAnimation . . . 85
8.6.2 ColorAnimation . . . 86
8.6.3 Transformatie animaties . . . 87
8.6.4 Animaties en XAML . . . 88
8.6.5 *C#*of XAML animaties? . . . 88
8.7 DependencyProperties in XAML: TypeConverters . . . 89

8.7.1 IValueConverter interface . . . 89
8.7.2 Default dataconverters . . . 90
8.7.2.1 TryParse . . . 90
8.8 Dependency properties en OnRender . . . 91
8.9 Veel voorkomende fouten . . . 91
8.10 Extra lectuur . . . 91
8.11 Dependency property: opgaves . . . 92

9 Templated controls **93**
9.1 Templated controls: duiding . . . 93
9.2 Relevante namespaces . . . 93
9.3 CaptionControl . . . 94
9.3.1 PARTS . . . 94
9.3.1.1 Caption control via PARTS: inleiding . . . 94
9.3.1.2 Caption Control via PARTS: uitwerking . . . 95
9.3.1.3 Caption control via Parts: test en aanpassing . . . 96
9.3.1.4 Caption Control via PARTS: evaluatie . . . 97
9.4 Caption Control opgave . . . 97
9.5 States . . . 97
9.6 State opgave . . . 99

10 Behaviors **101**
10.1 Behaviors: duiding . . . 101
10.2 Relevante namespaces . . . 101
10.3 Drag behavior . . . 102
10.3.1 Drag behavior: dependency property versie . . . 102
10.3.2 Drag behavior: Blend out of the box versie . . . 102
10.3.3 Drag behavior: gespecialiseerde Blend versie . . . 103
10.4 Grip behavior . . . 103
10.5 Resize behavior . . . 104
10.5.1 Alignment impact (Laurenz Demey 2010- 2011) . . . 104
10.6 Behavior opgaves . . . 105
10.7 WPF en application settings . . . 105
10.8 Enumerations: getalwaardes en bewerkingen . . . 107
10.9 Extra lectuur . . . 108

11 iTextSharp **109**
11.1 Installatie . . . 109
11.2 PDF creatie (en opgaves) . . . 109
11.3 Manipulatie van een bestaande PDF (en opgaves) . . . 110
11.4 Tekenen (en opgaves) . . . 110
11.5 Toepassingsgebied voor iText . . . 111

12 SQLServer en .NET integratie **113**
12.1 Nodige softwares . . . 113
12.2 Kwadraten berekenen . . . 114
12.3 SQL Server tabellen raadplegen . . . 115
12.4 Debugging SQLServer . . . 116
12.4.1 Debugging startend vanuit het Visual Studio project . . . 116
12.4.2 Debugging van een SQL Server call gestart buiten het project . . . 116
12.5 Theorie vragen . . . 117

13 Windows installer oplossingen **119**
13.1 Demo programma . . . 119
13.2 InstallShield Limited Edition . . . 120
13.2.1 Download en installatie . . . 120
13.2.2 Aanmaken van een MSI-pakket . . . 120
13.2.3 Redistributables . . . 120
13.2.4 Evaluatie InstallShield Limited Edition . . . 120
13.3 WiX . . . 121
13.3.1 Download en installatie . . . 121
13.3.2 Aanmaken van een MSI-pakket . . . 121
13.3.3 Redistributables . . . 121
13.3.4 Evaluatie WiX . . . 121
13.4 Welke installer? . . . 122
13.4.1 Vergelijking InstallShield en Wix . . . 122
13.4.2 Vergeet niet: . . . 122
13.5 Voorbeeld labo examen opgave . . . 122

14 AOP: Aspect Oriented programming **123**
14.1 AOP duiding . . . 123
14.1.1 Relevante namespaces . . . 123
14.2 PostSharp . . . 124
14.2.1 PostSharp kennismaking: methode-entry teller . . . 124
14.2.2 PostSharp en parameter annotaties . . . 125
14.2.3 PostSharp en performance counters . . . 127
14.2.3.1 Eigen performance counters . . . 128
14.2.3.2 AOP en performance counters . . . 128
14.3 AOP oefeningen . . . 129

15 Query Notification **131**
15.1 Query notification oefening . . . 131

16 Discussie topics **133**
16.1 Duiding . . . 133
16.2 Sinus berekenen mbv Taylor reeks . . . 134

17 Global Assembly Cache: GAC **135**
17.1 Global Assembly Cache: duiding . . . 135
17.1.1 Relevante namespaces . . . 135
17.2 GAC: technische duiding . . . 136
17.3 Ontwikkelen voor en met de GAC . . . 138
17.3.1 Strong named assemblies . . . 138
17.3.2 Gacutil.exe . . . 139
17.3.3 DLL-versies maken . . . 140
17.3.4 Redirecting assemblies . . . 141
17.3.4.1 Welke assembly wordt geladen? . . . 141
17.3.4.2 Redirecting met configuration file . . . 142
17.3.4.3 Redirecting met policy files . . . 143
17.3.4.4 Redirecting op machine niveau . . . 144
17.3.4.5 Redirecting redirections . . . 144
17.3.5 GAC en project references . . . 144
17.3.6 Installeren in de GAC vanuit de Setup (tem VS 2010) . . . 144
17.4 GAC opgave . . . 145
17.5 Extras . . . 146

18 Visual Studio geautomatiseerde broncode manipulaties **147**
18.1 Visual Studio geautomatiseerde broncode manipulaties: duiding . . . 147
18.2 Relevante namespaces . . . 147
18.3 Toolbox codeblocks . . . 148
18.3.1 Voordeel van toolboxcode . . . 148
18.3.2 Nadelen van toolboxcode . . . 148
18.4 Code snippets . . . 149
18.4.1 Snippets gebruiken . . . 149
18.4.2 Eigen code snippets . . . 150
18.4.3 Voordelen van snippets . . . 150
18.4.4 Nadeel van snippets . . . 150
18.5 Macro's in Visual Studio . . . 151
18.5.1 Macro samples . . . 151
18.5.2 Eigen macro's . . . 151
18.5.3 Voordelen van macro's . . . 151
18.5.4 Nadeel van macro's . . . 151
18.6 Visual Studio Addin's . . . 152
18.6.1 Creatie van een Visual Studio Add-in . . . 152
18.6.2 Testuivoering van een Add-in . . . 152
18.6.3 Een lijnenteller Add-In . . . 153
18.6.4 Reference historiek Add-In . . . 153
18.7 Visual Studio events . . . 155
18.8 Deployment van een Add-In . . . 156
18.9 Visual Studio en MEF: extentions . . . 157
18.9.1 cmnt extention . . . 157

18.9.2 cmt extention implementatie . . . 157
18.9.3 cmt deployment . . . 158
18.9.4 Extentions en Visual Studio . . . 158
18.10 Relevante literatuur . . . 158
18.11 Codemanipulatie opgaves . . . 159

Jef Daels

Hoofdstuk 1

Cursus aanpassingen

- templated controls: een no-code kalender (2 views: 3 cbo's, en een maandgrid met navigatie)
- content uit sexy $C^{\#}$

Jef Daels

Hoofdstuk 2

Random topics

2.1 Extention methods

Extention methods bieden de mogelijkheid om, op syntactische wijze, methodes toe te voegen aan klassen: u kan met behulp van de .- notatie deze methodes oproepen. Dit is enkel *syntactic sugar*, omdat de klasse definitie zelf niet wordt uitgebreid (hiervoor zou u overerving nodig hebben).

Het definiëren van een extention method gebeurd door middel van een static routine, zoals hieronder met een voorbeeld beschreven:

```
public static int AantalKlinkers(this string tekst)
    {
    int iAantal = 0;
    foreach (char c in tekst)
        if (IsKlinker(c)) iAantal++;
    return iAantal;
    }
```

Figuur 2.1: Extention method definition

Bij het coderen van een extention method zal de intellisense tonen dat het een extention method betreft:

```
public MainWindow()
    {
    InitializeComponent();
    Console.WriteLine("aantal klinkers in " + this.Title + ": "
        + this.Title.AantalKlinkers());
    int i = this.Title.aantal
    }
}
```

AantalKlinkers (extension) int string.AantalKlinkers()
All Tokens

Figuur 2.2: Extention method gebruik

2.1.1 Gebruik van Extention methods

2.1.1.1 LINQ en extention methods

Extention methods werden geïntroduceerd tesamen met LINQ, wiens implementatie grotendeels gebaseerd is op Extention methods (zie de intellisense bij het oproepen van een LINQ-methode)

2.1.1.2 Animaties en extention methods

Designers zullen animaties beschrijven door middel van XAML code (bijvoorbeeld bij een STATE-wijziging bij het gebruik van de VisualStateManager). Eenzelfde animatie zal op verschillende plaatsen gedupliceerd worden. Indien we de animatie als extention method implementeren kan ze eenvoudig (via code) herhaaldelijk worden toegepast:

```
spNumbers11.FadeTo((speler1Waits) ? 0.15 : 1.0, 500);
spNumbers12.FadeTo((speler1Waits) ? 0.15 : 1.0, 500);
spNumbers21.FadeTo((!speler1Waits) ? 0.15 : 1.0, 500);
spNumbers22.FadeTo((!speler1Waits) ? 0.15 : 1.0, 500);
```

Figuur 2.3: Extention method gebruik: fadeto

Opmerking: de gecodeerde animatie is gemakkelijk in een behavior te plaatsen zodat ze ook vanuit XAML snel kan worden geactiveerd.

2.1.2 Extention method surprises

Extention methodes gedefinieerd op een klasse zijn ook van toepassing op instanties van een ervende klasse.

2.2 Partial classes

2.3 Partial methods

2.4 Transformations

Een transformatie kan opgebouwd zijn uit verschillende elementen: een draaiing (rotatie) een schaling (scaling), een verplaatsing (translation) en kanteling (skewing). Indien u een tranformatie op een object toepast dient u rekening te houden met de reeds bestaande transformaties: indien u een object wenst te herschalen mag u de eventuele draaiing niet verwijderen. Een CompositeTransform object zal in veel gevallen uitermate geschikt zijn om met één object al uw tranformaties te definiëren en animeren. Dit vermijdt dat u voor een animatie alle TransformGroup onderdelen moet nakijken om de gepaste tranformatie te vinden.

2.5 Awaitable methodes

.NET 4.5 introduceert *awaitable* methodes. Deze methodes zijn herkenbaar aan de suffix *Async*. Hun uitvoering zal (normaal gezien :)) door het keyword *await* worden voorafgegaan. In dat geval zal de rest van de code pas worden uitgevoerd nadat de (asynchrone) oproep van de Async-methode is afgelopen (dit wordt meer in detail toegelicht in het hoofdstuk omtrent delegates en het hoofdstuk omtrent backtracking).

```
public static async Task XMLSerialise(Bord brd)
    {
    StorageFolder storageFolder = ApplicationData.Current.LocalFolder;
    StorageFile sf = await storageFolder.CreateFileAsync("numberlegions
    Stream s = await sf.OpenStreamForWriteAsync();
    XmlSerializer serializer = new XmlSerializer(typeof(Bord));
    serializer.Serialize(s, brd);
    s.Dispose();
    }
```

Figuur 2.4: Await demo

Belangrijk hierbij op te merken zijn volgende elementen:

- indien u een methode *await*, dan kan dit enkel in een methode die als *async* gedefinieerd werd;
- indien een methode als *async* gedefinieerd wordt is ze op haar beurt awaitable.
  - een async methode zal, indien opgeroepen zonder await, asynchroon worden uitgevoerd: de oproepende methode wacht niet op voltooiing vooraleer verder te gaan met zijn logica:

```
        async void brdNumbers_SpelerToggled(object se
            {
            _isToetsVerwerkingBusy = false;
            pSpeler1ScoreA.Text = "" + brdNumbers.Sco
            pSpeler2ScoreA.Text = "" + brdNumbers.Sco
            pSpeler1ScoreB.Text = "" + brdNumbers.Sco
            pSpeler2ScoreB.Text = "" + brdNumbers.Sco
            await Bord.XMLSerialise(brdNumbers);
            await ZetVolgendeZet(e);
            }
```

Figuur 2.5: Await demo async methode

2.6 (XML) serialisatie in .NET

aanbrengen In een context van application life cycle

2.7 Var keyword

als je type kent: gebruik het! var is vooral interessant om het resultaat type van een linq query niet zelf te moeten tikken.

2.8 TryParse methodes

boolean result, twee parameters, tweede parameter is ref, lijkt me zeer toepasbaar bij dataconverters in XAML

2.9 Dynamic keyword

2.9.1 lock en volatile keywords

2.9.2 WPF toolkit

2.9.3 Application settings in WPF

Best way to bind WPF properties to ApplicationSettings in C#?

2.9.4 conditional[debug]

2.9.5 custom build

2.9.6 sys tools

Jef Dacls

Hoofdstuk 3

Op aanvraag: Data Binding in WPF

Databinding biedt WPF de mogelijkheid om declaratieve visualisaties voor data te definiëren via XAML-code. Het betrachtte voordeel is het aanpasbaar maken van de voorstelling zonder de achterliggende programmeer code te moeten aanpassen. Dit introduceert bijna automatisch de mogelijkheid om verschillende templates op hetzelfde gegeven toe te passen. In het MVVM-model wordt deze manier van werken sterk gepromoot.

3.1 De klasse nMCTDocent

Dit hoofdstuk betreft het visualiseren van data door middel van bindings. Vooraleer we de bindings toelichten moeten we dan ook een dataklasse opzetten:

```
public class nMCTDocent
    {
    public String FNaam { get; set; }
    public String VNaam { get; set; }
    public string Foto { get; set; }

    public static ObservableCollection<nMCTDocent> Docenten
        {
        get
            {
            ObservableCollection<nMCTDocent> l = new ObservableCollection<nMCTDocent>();
            l.Add(new nMCTDocent() { FNaam = "Gates", VNaam = "Bill", Foto = @"fotos\bgates.png" });
            l.Add(new nMCTDocent() { FNaam = "Knuth", VNaam = "Donald", Foto = @"fotos\dknuth.jpg" });
            l.Add(new nMCTDocent() { FNaam = "Turing", VNaam = "Alan", Foto = @"fotos\aturing.png" });
            l.Add(new nMCTDocent() { FNaam = "Lovelace", VNaam = "Ada", Foto = @"fotos\alovelace.png" });

            return l;
            }
        }
```

Figuur 3.1: De klasse nMCTDocent

- het *ObservableCollection* type wordt verder in deze tekst toegelicht. Momenteel volstaat het om dit type te beschouwen als een soort *List*;

- het is belangrijk dat de docentlijst via een property wordt aangeleverd. Op deze manier zullen we wat verder in de tekst lijsten kunnen binden met de *Docenten*, waarbij elke docent op zijn beurt zal gebind worden aan zijn visualisatie.

3.2 DisplayMemberPath: the old way

Databinding bestond reeds voor het .NET tijdperk. Zo kon men de (tekst-) visualisatie van een data- element in een lijst koppelen aan een property- waarde van dit data-element. De koppeling gebeurde door het *DisplayMember* in te stellen op de lijst. In WPF bestaat deze mogelijkheid ook nog en moeten we het *DisplayMemberPath* voor de lijst instellen:

```
<ComboBox x:Name="cboDocenten" DisplayMemberPath="FNaam"
        Grid.Row="1" Grid.Column="2"  Margin="3" />
```

Figuur 3.2: DisplayMemberPath binding

Een foutieve property-naam bij het binden resulteert niet in een compilatie of runtime- fout: het gegeven wordt gewoon niet gevisualiseerd. U kan dit wel terugvinden in de output van uw programma (u merkt dat de binding case sensitive is):

```
System.Windows.Data Error: 40 : BindingExpression path error: 'fNaam' property not found on 'object' ''nMCTDocent'
System.Windows.Data Error: 40 : BindingExpression path error: 'fNaam' property not found on 'object' ''nMCTDocent'
System.Windows.Data Error: 40 : BindingExpression path error: 'fNaam' property not found on 'object' ''nMCTDocent'
System.Windows.Data Error: 40 : BindingExpression path error: 'fNaam' property not found on 'object' ''nMCTDocent'
```

Figuur 3.3: BindingPathError

- in bovenstaande situatie moet de invulling van de combobox elders ($C^{\#}$, de zogeheten *code behind*) gebeuren. Dit is de voornaamste tekortkoming van deze benadering waardoor de visualisatie sterk afhankelijk van het programmeer werk blijft;

- elk element in de combobox wordt ondervraagd om de waarde van zijn *FNaam* property te bekomen (u weet hoe dit kan via reflection);

- de bekomen waarde (een tekst) wordt getoond als voorstelling van het object. Eigenlijk ben ik verrast dat deze binding enkel teksten kan tonen (u zal verder in de tekst merken dat binding zeker niet beperkt is tot tekst). Onderstaande ILSpy informatie lijkt dit voor wat betreft de *DisplayMemberPath*- situatie echter te bevestigen:


```
// System.Windows.Controls.ItemsControl
private void UpdateDisplayMemberTemplateSelector()
{
    string displayMemberPath = this.DisplayMemberPath;
    string itemStringFormat = this.ItemStringFormat;
    if (string.IsNullOrEmpty(displayMemberPath) && string.IsNullOrEmpty(itemStringFormat))
    {
        if (this.ItemTemplateSelector is DisplayMemberTemplateSelector)
        {
            base.ClearValue(ItemsControl.ItemTemplateSelectorProperty);
        }
        return;
    }
    DataTemplateSelector itemTemplateSelector = this.ItemTemplateSelector;
    if (itemTemplateSelector != null && !(itemTemplateSelector is DisplayMemberTemplateSelector) && (base.ReadL
    {
        throw new InvalidOperationException(SR.Get("DisplayMemberPathAndItemTemplateSelectorDefined"));
    }
    this.ItemTemplateSelector = new DisplayMemberTemplateSelector(this.DisplayMemberPath, this.ItemStringFormat
}
```

Figuur 3.4: ILSpy DisplayMemberPath info

3.3 DataContext: default binding source

3.3.1 DataContext: duiding

Opdat een visualisatie een dataelement zou kunnen tonen moet het dit dataelement natuurlijk kennen. In een WPF- context spreken we dan van de *DataContext* van het UI-element. Een goed begrip van deze *DataContext* is cruciaal om databinding goed te begrijpen. Volgende punten zijn belangrijk:

- indien een UI-element zijn eigen datacontext niet expliciet instelt wordt de datacontext van de UI-parent overgenomen;
  - dit is de hoeksteen van het *MVVM*- gebeuren: we maken één static resource die we instellen als datacontext van het window, alle UI-elementen in dat window erven deze datacontext. Het is dan natuurlijk nodig dat elk viseel element een property-tegenhanger heeft in de viewmodel klasse;
  - uitzonderingen zijn items in lijsten, waarbij elke visualisatie van een lijst-element dit lijstelement als datacontext krijgt ingesteld (automatisch);
- een datacontext instellen zonder deze te gebruiken in een binding is geen enkel probleem;
- bij het ontwikkelen van mijn eigen templated controls zal ik in de constructor de control (bijna) altijd als zijn eigen datacontext instellen.

3.3.2 Datacontext: instellen op een window

3.3.2.1 ViewModel voor de demo

Ook nu moeten we een klasse (het viewmodel) opzetten om die later te gebruiken in de visualisatie:

```
public class BindingsDemoVM
    {
    public string WindowTitel {
        get { return "Binding demo"; }
        }
    public ObservableCollection<nMCTDocent> Docenten {
        get { return nMCTDocent.Docenten; }
        }
```

Figuur 3.5: ViewModel klasse

In eerste instantie levert het viewmodel twee properties:

- *WindowTitel* die gebind zal worden aan de Title- property van het Window;
- *Docenten* die een lijst van docenten zal aanleveren, die in een lijstcontrol (Combobox, ListView, GridView, ..) getoond kan worden.

3.3.2.2 Instellen van de Window DataContext

- in XAML moet een element reeds gedefinieerd zijn vooraleer het gebruikt kan worden. Omdat de resources van een Window na de *Window*- tag komen, en de Title- property in die Window- tag wordt ingesteld is het geen optie om de datacontext in de *Window.Resources* te definiëren. De plaats om de DataContext van het opstartwindow te definiëren is in de resources van de App.xaml- file (deze wordt gelezen voor het Window):

```
<Application.Resources>
    <src:BindingsDemoVM x:Key="vm" />
</Application.Resources>
```

Figuur 3.6: Application Resources

- op basis van de key kan dit (static) resource gebruikt worden in een binding:

```
<Window x:Class="BindingsDemo.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/pr
    Height="350" Width="525"
    Title="{Binding Path=WindowTitel}"
    DataContext="{Binding Source={StaticResource vm}}"
    >
```

Figuur 3.7: Window binding

De window- tag bevat twee bindings:

- het binden van de DataContext aan de static resource *vm*;
  - indien gewenst kan de DataContext ook via code in de constructor worden ingesteld. Dit kan zinnig zijn indien u geen DataContext instantie als resource kan aanmaken;
- het binden van de *Title* property van het window aan de *WindowTitel* property van de datacontext (indien geen source wordt ingesteld op de binding, dan is de datacontext de source);
  - indien de DataContext van de controls in het Window niet wordt overschreven kan nu elke String- property van elke control in het window (ook deze die elders in een template worden beschreven en nog geladen moeten worden) gebind worden aan de *WindowTitel* van de overgenomen datacontext (bemerk dat de datacontext niet expliciet wordt ingesteld):

```
<TextBlock Text="{Binding WindowTitel}" Grid.Row="3" Grid.Column="1" />
```

Figuur 3.8: DataContext binding

3.3.2.3 Binden van de ItemsControl.ItemsSource Property

Indien we lijsten wensen te visualiseren zullen we hiervoor wellicht de ItemsControl klasse (of een afgeleide) gebruiken. Deze klasse heeft een *ItemsSource* property van het (Interface-) type IEnumerable: dit kan zowel een List, een Array, .. zijn, zolang de IEnumerable interface maar geïmplementeerd wordt:

ItemsControl.ItemsSource Property

.NET Framework 4.5 | Other Versions | 0 out of 5 rated this helpful

Gets or sets a collection used to generate the content of the ItemsControl.

Namespace: System.Windows.Controls
Assembly: PresentationFramework (in PresentationFramework.dll)
XMLNS for XAML: http://schemas.microsoft.com/winfx/2006/xaml/presentation, http://schemas.

Syntax

C# | C++ | F# | VB

```
[BindableAttribute(true)]
public IEnumerable ItemsSource { get; set; }
```

Figuur 3.9: ItemsSource declaratie

Indien u in een $C^{\#}$- window *F12* drukt op het *ObservableCollection*- type wordt de gekende code getoond:

```
public class ObservableCollection<T> : Collection<T>, INotifyCollectionChanged, INotifyPropertyChanged
```

Figuur 3.10: Observable hoofding

Indien we nu terug *F12* duwen op *Collection* dan bekomen we:

```
public class Collection<T> : IList<T>, ICollection<T>, IList, ICollection, IReadOnlyList<T>, IReadOnlyCollection<T>, IEnumerable<T>, IEn
```

Figuur 3.11: Collection hoofding

Hieruit kunnen we afleiden dat de *ObservableCollection*- klasse, via het erven van de *Collection*- klasse ook de *IEnumerable* interface implementeert, en de *Docenten* property van het modelview dus geschikt is als bindings source van de ItemsSource van een ComboBox:

```
<ComboBox x:Name="cboDocentenBound" Grid.Row="2" Grid.Column="2"
          DisplayMemberPath="FNaam" Margin="3"
          ItemsSource="{Binding Docenten}" />
```

Figuur 3.12: ItemsSource binding

Opdat de combobox zou weten welke informatie van het dataelement getoond moet worden wordt ook nu terug de *DisplayMemberPath* property ingesteld.

3.3.2.4 Instellen van de ItemsControl.ItemTemplate Property

Om een rijke user experience mogelijk te maken willen we natuurlijk meer dan enkel tekst waardes kunnen tonen. Indien u een DataTemplate hebt voor uw te tonen dataelement kan u dit in de *ItemsControl* instellen met behulp van de *ItemTemplate* property die dan wordt toegepast op elk van de items in de *ItemsSource*:

```
<ComboBox x:Name="cboDocentenBoundTemplated" Grid.Row="3" Grid.Column="2"
          Margin="3"
          ItemTemplate="{StaticResource tmplDocenten}"
          ItemsSource="{Binding Docenten}" />
```

Figuur 3.13: ItemTemplate instellen

Bovenstaande code veronderstelt het bestaan van de *tmplDocenten* datatemplate. Deze wordt gedefinieerd in een resource sectie van bijvoorbeeld het window of de app.xaml file:


```
<DataTemplate x:Key="tmplDocenten" DataType="src:nMCTDocent" >
    <Grid Margin="4">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" SharedSizeGroup="Key" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <TextBlock Text="{Binding FNaam}" FontWeight="Bold"  />
        <Image Source="{Binding Foto}" Grid.Column="1" />
    </Grid>
</DataTemplate>
```

Figuur 3.14: nMCTDocent datatemplate

Het resultaat van deze binding met deze template ziet er als volgt uit:

Figuur 3.15: nMCTDocenten in een ComboBox

3.4 Niet DataContext binding sources

De DataContext, zoals de naam reeds suggereert, laat toe om in een binding de gegevens van het te visualiseren element (de data) te gebruiken. Deze datacontext is niet altijd de bron van het nodige gegeven. Voorbeelden waar de nodige informatie elders gehaald moet worden zijn:

- in de ccCaption control worden images getoond. We wensen deze als vierkant te tonen, wiens breedte gelijk is aan de hoogte van de Grid waarin ze worden geplaatst;
- bij het ontwikkelen van een custom control wensen we de properties die de designer instelt op de control te gebruiken in de template;

Om bovenstaande gevallen te kunnen afhandelen moet de *binding source* ingesteld worden. Dit kan gebeuren aan de hand van één van drie properties:

- Source: indien u een bepaalde property wenst te binden aan een ander (static) resource dan de huidige datacontext, zonder de datacontext aan te passen

- RelativeSource (MSDN): om de binding source aan te geven relatief ten opzichte van de control die de binding uitvoert. De syntax voor deze source instelling steunt op RelativeSourceMode Enumeration. Googlen zal ook hier uw beste vriendje zijn (lees zeker ook de MSDN entry);
- ElementName: indien u de binding source die u wenst te gebruiken bij naam kan identificeren in de Visual Tree.

Wanneer bijvoorbeeld de breedte van een GridColumn gebind moet worden aan de hoogte van die Grid kan onderstaande XAML-code gebruikt worden:

```
<DataTemplate x:Key="tmplDocenten2" DataType="src:nMCTDocent" >
    <Grid Margin="4" Height="60">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" SharedSizeGroup="Key" />
            <ColumnDefinition
                Width="{Binding RelativeSource={RelativeSource FindAncestor,
                AncestorType={x:Type Grid}}, Path=ActualHeight}"/>
        </Grid.ColumnDefinitions>
        <TextBlock Text="{Binding FNaam}" FontWeight="Bold" Style="{StaticResource txbstyle}"  />
        <Image Source="{Binding Foto}" Grid.Column="1" />
    </Grid>
</DataTemplate>
```

Figuur 3.16: RelativeSource binding

3.5 TypeConverter

Zie TypeConverter

Hoofdstuk 4

Delegates

4.1 Delegates: duiding

In een volledig object georiënteerde programmeeromgeving is het mogelijk om code- onderdelen als objecten te aanzien. Dit betekent dat deze onderdelen (routines, classes, assemblies, types, ..) instanties zijn van klasses die dergelijke code elementen beschrijven.

Routine- objecten (het onderwerp van dit hoofdstuk) zijn instanties van een *Delegate-type*. Deze Delegate- types zijn klasses die erven van de *Delegate*- class. Deze laatste is zelf geen Delgate-type (er zijn geen routines die instanties zijn van de basis Delegate-class).

In het hoofdstuk omtrent reflection zullen we ook andere code onderdelen als objecten verwerken. Routines zullen daar ook als instanties van (Reflection-) classes terugkeren. In een reflection context zullen deze instanties vooral beschrijvende (meta-) informatie bevatten.

Delegates spelen in .NET een belangrijke rol bij het implementeren van events, multithreading en codeplumbing. Ook de implementatie van WCF en (attached) dependency properties maken gebruik van delegates. Delegates zijn dan ook zonder twijfel een belangrijke werkstuk voor de betere programmeur!

4.1.1 Relevante namespace

- *System*

4.2 Delegates: technische intro

4.2.1 Demo-code en toelichting

De meest voor de hand liggende actie die we op een routine willen toepassen, is deze uitvoeren. Hierbij is het van belang te weten welke argumenten moeten voorzien worden en van welke type het eventuele resultaat een instantie zal zijn. Een delegate- type definitie zal zich hiertoe beperken: welke zijn de argumenttypes, en welk is het eventuele resultaat type? De naam van de routine en de parameters, en de geïmplementeerde logica, zijn volledig irrelevant voor de delegate type definitie. Een voorbeeld delegate- type definitie:

```
public delegate void DoeIetsMetTekstDelegate(string sTekst);
```

Figuur 4.1: Delegate definitie

Merk hierbij volgende punten op:

- het keyword *delegate* heeft aan dat we een delegate-type definiëren (een speciale klasse, zonder body);
- het return type wordt genoteerd voor de naam van het delegate type. In het voorbeeld is dit return type void (we verwachten geen resultaat);
- de argumentnamen zijn niet van belang (en moeten zeker niet overeenstemmen met de argumentnamen gebruikt in de routines die instanties van deze klasse zullen zijn). De argumenttypes zijn *wel* belangrijk!

```
public delegate void doeIetsMetTekst(String tekst);

public partial class MainWindow : Window
    {
    private void btnDemoUCLC_Click(object sender, RoutedEventArgs e)
        {
        doeIetsMetTekst routine = new doeIetsMetTekst(toonUC);  //maak een routinereferentie en koppel ze
        routine.Invoke("hallo");    //start de methode gekoppeld aan deze routinereferentie, en voorzie de oproep van de nodige parameters
        routine = new doeIetsMetTekst(toonLC);
        routine.Invoke("hallo");
        }
    private void toonUC(string tekst) { MessageBox.Show(tekst.ToUpper()); }
    private void toonLC(string tekst) { MessageBox.Show(tekst.ToLower()); }
```

Figuur 4.2: Delegate demo: uppercase en lowercase

Bovenstaande demo-code zal u wellicht niet overtuigen van het belang van delegates: het aanroepen van een methode wordt gewoon complexer gemaakt. Verder in dit hoofdstuk behandelen we een aantal (niet exhaustief (=uitputtend)) situaties waarbij het gebruik van delegates een elegante oplossing biedt.

Een delegate- definitie is niet hetzelfde als een signatuur definitie: de routinenaam maakt onderdeel uit van de signatuur (en niet van het delegate- type), en het returntype is geen onderdeel van de signatuur, maar wel van belang in de delegate definitie.

4.2.2 Covariance en contravariance

Wat indien een routine hoofding niet volledig overeenstemt met de delegate definitie? Een routine kan op twee manieren afwijken van een delegate type definitie, en toch nog als instantie van dit delegate type worden aanzien: covariance en contravariance. Beide worden beschreven in Covariance and Contravariance in Delegates (C# Programming Guide). Een combinatie van beide is natuurlijk ook toegelaten.

4.2.2.1 Covariance

Indien het returntype van een routine een subclasse is van het return type van het delegate type, dan is deze routine toch een instantie van dit delegate type. Uiteraard levert dit geen problemen op, gezien elk resultaat kan gecast worden naar het return type van het delegate type.

4.2.2.2 Contravariance

Indien een argument van de routine minder gespecialisserd is dan het overeenkomstige argument in het delegate type is er uiteraard ook geen probleem: elke actuele parameter moet voldoen aan het delegate (formele) parameter type, en voldoet dus zeker ook aan het algemenere routine parameter type.

4.3 Delegates en events

Events leveren een mechamisme waarbij een server-object (levert diensten) initiatief neemt om code door een client-object te laten uitvoeren. In .NET wordt dit gerealiseerd door middel van delegates (dit in tegenstelling tot JAVA waar men gebruikt maakt van interfaces (denk bijvoorbeeld aan de *ActionListener* interface)).

Het client of server zijn van een object ten opzichte van elkaar hangt af van de te leveren dienst: indien we een Button op een Window plaatsen dan is deze Button client van het Window omwille van de visuele ondersteuning (het Window levert een dienst), en is het Window evenzeer client van de Button omdat de Button onder andere een click- dienst aanlevert. Het is dus zeker niet zo dat een client/server relatie gelezen moet worden als een klein/groot- relatie.

4.3.1 Definitie van een event

De definitie van een event voorziet een delegate- typed field van het keyword *event*:

```
public delegate void EventVoorbeeldNietVolgensGuidelines(string tekst); //delegate voor het event
public class DemoTekstVak : TextBox
    {
    public event EventVoorbeeldNietVolgensGuidelines TekstIsVijfLang; //eventdefinitie
    public DemoTekstVak()
        {
```

Figuur 4.3: Event definitie (zonder guidelines)

De delegate in de eventdefinitie beschrijft de client-routines die aan dit event gekoppeld kunnen worden. Bemerk dat de code in dit voorbeeld *niet* voldoet aan de guidelines omtrent eventhandling.

Bovenstaande code is onderdeel van een gespecialiseerde TextBox klasse die voorzien wordt van een TekstIsVijfLang event. Het is de bedoeling dit event af te vuren indien de tekst in het tekstvak exact 5 tekens lang wordt.

4.3.2 Abonneren (to subscribe) op een event

De syntax nodig om een client te abonneren op een event is verschillend met deze van VB.NET (waarbij een *handles* clause achteraan een routine werd toegevoegd). In $C^{\#}$wordt een routine gekoppeld aan een event door middel van de *+=*- operator:

```
public MainWindow()
    {
    InitializeComponent();
    txt5Tekens.TekstIsVijfLang += new EventVoorbeeldNietVolgensGuidelines(txt5Tekens_TekstIsVijfLang);
    }
void txt5Tekens_TekstIsVijfLang(string tekst)
    {
    Console.WriteLine(tekst + " is 5 tekens lang");
    }
```

Figuur 4.4: Event abonnering

Visual Studio ondersteunt het abonneren op volgende wijze:

- indien u na de *+=*- operator de *TAB* toets indrukt wordt automatisch een correcte right-hand expressie gegenereerd;
- indien u hierna nog eens *TAB* indrukt zal ook een correcte eventhandler routine gegenereerd worden, rekening houdend met de event definiërende delegate.
- indien u dubbelklikt op een control wordt er een *+=*- expressie aangemaakt in de generated code en wordt een passende eventhandler in de *.cs*- fiel geplaatst;

4.3.2.1 Event subscription: verschil met VB.NET

- Handles- clause: Hiervoor bestaat geen $C^{\#}$- equivalent: $C^{\#}$kent enkel de *+=*- operator om routines aan events te koppelen;

- AddHandler: De VB.NET handles clause is niet in staat om routines *at run-time* aan events te koppelen. Hierdoor is deze werkwijze ongeschikt om events van bijvoorbeeld dynamisch toegevoegde controls op te vangen. VB.NET kent de *AddHandler* instructie om at run-time routines aan events te koppelen. Indien u de documentatie erop naleest merkt u dat VB.NET de *AddressOf*- operator gebruikt in combinatie met de AddHandler instructie.

4.3.3 Afvuren van een event

De klasse die het event definieert kan dit event ook uitvoeren. Het is belangrijk om te testen of er clients zijn voor dit event (indien er geen clients zijn is het field *null* en zou het afvuren een uitvoeringsfout veroorzaken):

```
public delegate void EventVoorbeeldNietVolgensGuidelines(string tekst); //delegate voor het event
public class DemoTekstVak : TextBox
    {
    public event EventVoorbeeldNietVolgensGuidelines TekstIsVijfLang; //eventdefinitie
    public DemoTekstVak()
        {
        this.TextChanged += new TextChangedEventHandler(DemoTekstVak_TextChanged);
        }

    void DemoTekstVak_TextChanged(object sender, TextChangedEventArgs e)
        {
        if (this.Text.Length == 5)
            if (TekstIsVijfLang != null)    //minstens 1 client voor dit event nodig, anders null
                TekstIsVijfLang(this.Text); //starten van het event (voor alle clients)
        }
    }
```

Figuur 4.5: Event afvuren (zonder guidelines)

Bemerk dat in bovenstaande code, die het afvuren van een event illustreert, er ook een event abonnering aanwezig is: het object abonneert zichzelf (in de constructor) op zijn eigen *TextChanged*- event (door middel van de *+=- operator*) om na te kijken of de tekst na wijziging exact vijf tekens lang is.

4.3.3.1 Afvuren van een event: verschil met VB.NET

- *RaiseEvent* is de instructie om in VB.NET een event af te vuren. Het is *niet* nodig om te controleren of er clients zijn voor dit event.

4.3.4 Meerdere clients voor hetzelfde event

Het is perfect mogelijk om meerdere event-handlers te koppelen aan eenzelfde event. Bij het afvuren van het event zullen de gekoppelde routines in volgorde, één voor één uitgevoerd worden.

4.4 Event Guidelines!

Hoewel er technisch geen beperkingen worden opgelegd aan de event definiërende delegates worden door Microsoft volgende guidelines beschreven:

- het return type is *void*;
- de delegate heeft exact twee argumenten:
  - Object sender: het object waardoor het event wordt gestart;
  - EventArgs e: EventArgs mag vervangen worden door een klasse die overerft van EventArgs indien u extra informatie aan het event wenst toe te voegen (voorbeelden hiervan zijn de mouselistener events);
- het afvuren van het event gebeurt in een overschrijfbare (*protected virtual*) methode. Dit laat ervende klasses toe om het event gedrag aan te passen (zie DemoButton in het demo project). De methodenaam is *onEvent* waarbij Event natuurlijk wordt vervangen door de naam van het event.
- gebruik *System.EventHandler<T>* in plaats van zelf aangemaakte delegate definities.

Deze guidelines *moeten* in deze cursus worden toegepast opdat uw event-implementaties als correct zouden worden aanzien.

Als een eerste oefening kan men de event implementatie, zoals hierboven beschreven, hercoderen zodat ze voldoet aan de guidelines.

Wie niet (voldoende) bekend is met *generics* (*System.EventHandler<T>*) wordt verondersteld dit op te zoeken. Indien zelfstudie onvoldoende duidelijkheid schept kan u vragen stellen in de komende lessen.

4.5 Omtrent Events:

4.5.1 Static events

Meestal associëren we een event met een instantie. Om te abonneren op een event van een instantie hebben we een referentie naar die instantie nodig. Dit is niet altijd mogelijk: veronderstel dat client code wenst te reageren op het aanmaken van een nieuw object van een klasse: het is niet mogelijk in te tekenen op het *PersoonCreated* event van een nog onbestaand object. .NET voorziet de mogelijkheid om static events te definiëren: de client abonneert dan op een (static) event van een klasse:

```
public class clsPersoon
    {
    public static event EventHandler<PersoonCreatedEventArgs> PersoonCreated;
    public string FNaam { get; set; }   //property definitie
    public string VNaam { get; set; }   //property definitie
    public clsPersoon() : this("","")
        {
        }
    public clsPersoon(string fnaam, string vnaam)
        {
        FNaam = fnaam;
        VNaam = vnaam;
        if (PersoonCreated != null)
            PersoonCreated(null, new PersoonCreatedEventArgs(this));
        }
    public override string ToString()
        {
        return this.FNaam + ", " + this.VNaam;
        }
    }
public class PersoonCreatedEventArgs : EventArgs
    {
    private clsPersoon _persoon;
    public PersoonCreatedEventArgs(clsPersoon persoon) { _persoon = persoon; }
    public clsPersoon persoon { get { return _persoon; } }
    }
```

Figuur 4.6: Static event definitie (volgens de guidelines)

```
public MainWindow()
    {
    InitializeComponent();
    clsPersoon.PersoonCreated += new EventHandler<PersoonCreatedEventArgs>(clsPersoon_PersoonCreated);
    }

void clsPersoon_PersoonCreated(object sender, PersoonCreatedEventArgs e)
    {
    Console.WriteLine(e.persoon.ToString());
    }
```

Figuur 4.7: Static event client

4.5.2 Delegates of Interfaces: .NET of JAVA events

In .NET gebruikt men delegates om eventhandling te realiseren. In JAVA definieert men interfaces om eventhandling mogelijk te maken: een instantie van een interface implementerende klasse kan dan als eventhandler aan een object toegekend worden. Welke is te verkiezen?

Bij manuele codering (wie doet dat nog?) moet in JAVA elk interface element voorzien worden, ook al bent u enkel geïnteresseerd in een specifieke routine. In .NET moet u enkel die events die u interesseren van code voorzien. Beide oplossingen zijn voldoende eenvoudig om een intuitief gebruik mogelijk te maken (zeker indien de IDE u ook nog wat helpt). Beide benaderingen zijn dan ook sterk gelijkwaardig: ik verkies er geeneen boven de andere.

4.5.3 Routed events

Voorgaande eventhandling is reeds van bij de start van .NET aanwezig in het framework. Recentelijk (WPF, .NET 3.5) werden ook *routed events* geïntroduceerd. Deze events zijn WPF specifiek en worden geïntroduceerd om volgende problemen op te lossen:

- veronderstel een Button, met daarop een tekening (Image);
- de gebruiker klikt op de tekening: wellicht wil hij de button inklikken? Misschien ook niet? Hoe wordt de Button op de hoogte gesteld van het klikken op de Image? Wat indien er geen Image is?

Een routed event is een event (anders dan voorgaande) om binnen een geneste lijst van controls een gebeurtenis door te geven. We spreken van *tunneling events* indien de container het event doorgeeft naar zijn child, en van *bubbling events* indien het child het event doorgeeft aan zijn container. Een parent (bijvoorbeeld Window)is in staat om te subscriben (addHandler) op het routed event (bijvoorbeeld *TextChanged*- event van een bij abonnering niet gekend child (zie ook Sudoku routed events).

Indien een WPF- gebeurtenis zich voordoet zal het doorgeven van events in twee stappen gebeuren:

- eerst wordt het event van de containers naar de children doorgegeven: *tunneling*. Deze events worden prefixed met *preview*;
- daarna volgt een bubbling fase waarbij de gewone event naam wordt gebruikt;
- indien een control meent het event correct af te handelen kan de handled property van het tweede event-argument op *true* worden geplaatst, wat het doorgeven van die specifieke fase (grotendeels) stopt.

Interessant om weten:

- routed events bestaan enkel in een WPF-omgeving. Ze kunnen dan ook niet gebruikt worden om eventhandling buiten de WPF omgeving (uw visualisatie) te implementeren.
- snoop is een leuke tool om WPF-constructies te onderzoeken. Ook het verloop van routed events kan met deze tool worden nageplozen.

4.6 Delegates en multithreading

4.6.1 Multithreading: wat bedoelt u?

Bij het uitvoeren van een programma worden instructies één voor één door de CPU verwerkt. Een thread is een onderdeel van een proces (programma) dat in staat is om opeenvolgende instructies uit te voeren. Een multithreaded programma heeft meerdere onafhankelijk van elkaar uitvoerende threads. Wanneer het programma wordt uitgevoerd door een multithreaded operating system worden verschillende zaken op hetzelfde moment uitgevoerd.

4.6.2 Multithreading: iets voor ons?

Multithreaded programmeren is *ongeloofelijk* delicaat: indien gelijktijdige threads eenzelfde variabele wensen aan te passen krijgen we de problemen die we kennen vanuit de cursus databases: *lost update*, *inconsistent read* en *ghost data* om er slechts enkele te noemen. Het schrijven van correcte multithreaded toepassigen is dan ook geen *walk in the park/piece of cake*.

Betekent dit dat we geen multithreading kunnen programmeren? Natuurlijk niet: het ophalen van een grote lijst gegevens, het verwerken van een lijst gegevens die niet gewijzigd worden: allemaal ideale kandidaten om multithreaded uitgewerkt te worden. Indien er zich concurrency (gelijktijdigheid) problemen kunnen voordoen zullen we locks gebruiken. Bij het database programmeren worden locks door het RDBMS geplaatst. In *C#*(en VB.NET) kunnen we dit zelf programmeren met behulp van de *lock*- instructie.

Wanneer we spreken over multithreaded toepassingen hebben we het veelal ook over:

- parallel processing: parallelle (gelijktijdige) uitvoeringen van meerdere threads impliceren multithreading;
- asynchrone verwerking (in tegenstelling tot synchrone verwerking bij single threaded toepassingen): hierbij wordt een routine gestart en wachten we niet op het einde van deze routine vooraleer we verder gaan. Asynchrone verwerking wordt in .NET gerealiseerd met behulp van delegates. Een voor de hand liggend probleem dat moet worden opgelost: hoe weet mijn programma dat een asynchrone uitvoering afgelopen is (om bijvoorbeeld een *printen beëindigd* boodschap te kunnen tonen?

Multithreaded applicaties zijn ook op een single- core CPU belangrijk: zo kan men het printen van een document op de achtergrond laten uitvoeren, terwijl de user-responsiveness niet daalt. Met de introductie van multi-core CPU's wordt het ontwikkelen van multi-threaded toepassingen nog interessanter: een single-threaded toepassing zal op een quad core nooit meer dan vijventwintig procent van de beschikbare rekenkracht kunnen gebruiken. Dit maakt het multi-threaded zijn van uw toepassing belangrijk indien u het volle potentieel van de machine wenst te gebruiken.

4.6.3 Asynchrone processing en delegates

4.6.3.1 Asynchroon, geen AsyncCallback

Bij een synchrone verwerking zal een routine de volledige utivoering van een instructie (eventueel een deelprobleem) afwachten vooraleer de volgende instructie wordt gestart. Zo zal de uitvoering van btnDemoUCLC_click in Delegate demo: uppercase en lowercase na het tonen van de eerste messagebox wachten tot de gebruiker op *ok* klikt vooraleer de tweede messagebox wordt getoond.

Wanneer we de code aanpassen door de *invoke/1*- methods van de delegates te vervangen door *BeginInvoke/3* dan worden de delegate routines asynchroon ten opzichte van de oproepende methode uitgevoerd:

```
private void btnAsyncDemo_Click(object sender, RoutedEventArgs e)
    {
    doeIetsMetTekst routine = new doeIetsMetTekst(toonUC);  //maak een routinereferentie en koppel ze
    routine.BeginInvoke("hAllO", null, null);    //start de delegate asynchroon
    routine = new doeIetsMetTekst(toonLC);
    routine.BeginInvoke("hAllO",null,null);
    }
```

Figuur 4.8: Asynchrone verwerking demo

Op te merken bij de nieuwe situatie:

- deze uitvoering resulteert in drie threads: de thread die het klik- event afhandelt, en een tread voor elk van de asynchrone oproepen;
- *BeginInvoke/3* kent drie parameters in plaats van één: de twee laatste parameters van een asynchrone delegate oproep bevaten een callback routine (de routine die wordt uitgevoerd als de asynchroon gestartte routine eindigt) en een status object als laatste argument (beide komen in het volgende punt aan bod). Het is de bedoeling om in dit status object informatie, verzameld gedurende de asynchrone uitvoering (bijvoorbeeld het goed of slecht aflopen) terug te geven aan de oproeper. Dit laatste argument is (later) terug te vinden als state property van de *IAsyncResult*- parameter van de *AsyncCallback* routine.
- de callback methode wordt via een AsyncCallBack- delegate aan de asynchrone oproep meegegeven (de callback methode is die routine die zal worden uitgevoerd wanneer de asynchrone oproep afgelopen is). Het is belangrijk op te merken dat de callback routine wordt uitgevoerd op de asynchrone thread!

4.6.3.2 Asynchroon, met AsyncCallback, zonder UI

```
private void btnAsyncDemo_Click(object sender, RoutedEventArgs e)
    {
    doeIetsMetTekst routine = new doeIetsMetTekst(toonUC);  //maak een routinereferentie en koppel ze
    routine.BeginInvoke("hAllO", new AsyncCallback(asyncAfgelopen), this);    //start de delegate asynchroon
    routine = new doeIetsMetTekst(toonLC);
    routine.BeginInvoke("hAllO",null, null);
    }
private void asyncAfgelopen(IAsyncResult iar)
    {
    toonResultaat(iar);
    }
private void toonResultaat(IAsyncResult iar)
    {
    object state = iar.AsyncState;  //bevat laatste argument van de oproep
    this.Title = "afgelopen";
    }
```

Figuur 4.9: AsyncCallback demo

In bovenstaande figuur zal de eerste asynchrone uitvoering bij het aflopen de routine *asyncAfgelopen* uitvoeren (op de nieuwe thread). In het *IAsyncResult* argument vinden we informatie omtrent de uitgevoerde routine. De callback routine wordt uitgevoerd op de nieuw gestarte thread. Bovenstaande stukje code loopt fout omdat we op de nieuwe thread de UI wensen aan te passen:

Figuur 4.10: UI-thread problem

In een Windows-omgeving is het enkel mogelijk om de UI-elementen aan te passen op de thread die deze UI-elementen aanmaakte. Indien we in de UI feedback omtrent een afgelopen asynchrone methode wensen te zien, zullen we in staat moeten zijn om naar de UI-thread terug te keren. U raadt het al: dit gebeurt met behulp van delegates.

4.6.3.3 Asynchroon, met AsyncCallback, met UI

```
private void btnAsyncDemo_Click(object sender, RoutedEventArgs e)
    {
    doeIetsMetTekst routine = new doeIetsMetTekst(toonUC);  //maak een routinereferentie en koppel ze
    routine.BeginInvoke("hAllO", new AsyncCallback(asyncAfgelopen), this);    //start de delegate asynchroon
    routine = new doeIetsMetTekst(toonLC);
    routine.BeginInvoke("hAllO",null, null);
    }
private void asyncAfgelopen(IAsyncResult iar)
    {
    Dispatcher.BeginInvoke(new AsyncCallback(toonResultaat),iar);   //Terug naar de ui thread
    }
private void toonResultaat(IAsyncResult iar)
    {
    object state = iar.AsyncState;  //bevat laatste argument van de oproep
    this.Title = "afgelopen";
    }
```

Figuur 4.11: Dispatcher demo

Het terugkeren naar de UI-thread gebeurt afhankelijk van de UI-technologie:

- WPF: elke visueel element heeft een *Dispatcher* property. Deze kan een thread starten op zijn UI-thread door middel van de *BeginInvoke/2* methode. Hoewel deze methode dezelfde naam draagt als deze die de delegate routines asynchroon start is het een andere methode (geen twee extra argumenten bijvoorbeeld);
- WinForms: elke control heeft een *BeginInvoke* methode. Deze start een delegate op de thread die de control maakte (de UI-thread).

Het is interessant op te merken dat een *Messagebox.show* oproep zichzelf terug op de gepaste thread plaatst.

4.6.4 Multithreading met de Thread klasse

Een alternatief om multithreaded toepassingen te maken kan gebruik maken van de *Thread*-klasse. Dit valt buiten de scope van deze les.

Jef Daels

4.6.5 Event based Ansynchronous Pattern (EAP)

Het Event based Asynchronous Pattern plaatst het asynchroon ontwikkelen in de alomgekende event- context:

- zowel de langlopende taak (op de andere thread) als de callback worden gekoppeld aan events (zie het BackgroundWorker voorbeeld hieronder);
- op basis van de uitleg omtrent events zou het duidelijk moeten zijn dat dit een wrapper-oplossing rond de asynchroon uit te voeren delegate en zijn callback is.
- het pattern voorziet ook de mogelijkheid om een langdurende bewerking te onderbreken en om voortgangs events af te vuren. U vindt dit terug in de documentatie :).

De BackgroundWorker class is een mooi voorbeeld van deze benadering:

- het DoWork- event: ik kan het niet beter uitleggen dan wat u in MSDN vindt. Bemerk de relatie met RunWorkerAsync
- het RunWorkerCompleted- event wordt ook netjes uitgelegd op MSDN.

4.6.6 Asynchroon programmeren made easy

Voorgaande code, waarbij het nodig was om *Callback delegates* en *DispatcherThreads* te begrijpen maakte het niet eenvoudig om multithreaded toepassingen op te zetten. In .NET 4.5 werd gepoogd om dit (syntactisch) een flink stuk te vereenvoudigen met de introductie van de *await*- en *async* keywords:

```
public async Task XMLSerialise()
    {
    //dit wordt enkel gebruikt als er nog geen bord geserialiseerd was (su
    StorageFolder storageFolder = ApplicationData.Current.LocalFolder; //a
    StorageFile sf = await storageFolder.CreateFileAsync("numberlegionsspl
    Stream s = await sf.OpenStreamForWriteAsync();
    XmlSerializer serializer = new XmlSerializer(typeof(SplashPage));

    serializer.Serialize(s, this);
    s.Dispose();
    }
```

Figuur 4.12: Await- keyword

- het *await*- keyword impliceert volgende zaken:
  - de oproep rechts van het *await*- keyword wordt asynchroon uitgevoerd;
  - de rest van de methode wordt slechts uitgevoerd nadat de asynchrone oproep eindigt. De compiler zal hiertoe de rest van de methode in een delegate wrappen en als callback met de asynchrone oproep meegeven;

- het *async*- keyword in een routine hoofding heeft aan dat deze routine *awaited* kan opgeroepen worden. Meer nog, indien een routine een *awaited*- oproep bevat (typisch een methode oproep die lang kan duren) moet deze routine verplicht *async* gedefinieerd worden. Dit maakt het mogelijk (niet noodzakelijk) om deze routine zelf ook terug *awaited* op te roepen;
  - indien u een *async* methode *niet* awaited oproept dan wordt ze synchroon uitgevoerd (en blokkeert ze dus de oproepende thread).
  - async methodes moeten een return type *Task* (indien geen resultaat- waarde) of *Task<resultaattype>* indien een resultaat wordt terug gegeven.
- bij het ontwikkelen van *Modern Style* apps (Microsoft tablet toepassingen) bent u verplicht deze keywords te gebruiken: het manipuleren van files, het raadplegen van webservices: al deze functionaliteit is enkel nog in een asynchrone versie beschikbaar;
- wat u momenteel nog ontbreekt is de kennis om zelf een asyn methode te ontwikkelen (zonder gebruik te maken van een andere async methode die u await). Dit zal aan bod komen in het hoofdstuk omtrent backtracking waar we onze artificiële intelligentie asynchroon zullen oproepen.
  - voor wie hierop niet kan wachten: we zullen gebruik maken van de TaskCompletionSource klasse.

Jef Daels

4.6.7 CancellationToken

De CancellationToken structure wordt gebruikt in GeolocationCS sample. Dit token dat bij het starten van een Task wordt meegegeven geeft de caller de mogelijkheid om de Task af te breken. Hiertoe is het nodig dat de gestarte Task regelmatig dit token raadpleegt en indien nodig zijn bewerkingen afbreekt.

4.6.8 Thread.Sleep

Indien meerdere threads met dezelfde prioriteit uitgevoerd moeten worden kan het interessant zijn om de uitvoerende thread te onderbreken zodat de andere thread met dezelfde prioriteit aan bod kan komen. Hiertoe kan u gebruik maken van de (statische) Thread.Sleep - methode. U zal dit nodig hebben om de multithreaded teller oefening op te lossen.

4.6.9 Meer info?

Over multithreading zijn boeken, wellicht ganse bibliotheken, geschreven. De laatste jaren is er hernieuwde aandacht voor deze topic omwille van de multi core processoren (de programmeer talen worden aangepast om parallelle verwerking gemakkelijker te ondersteunen). Een eventueel startpunt om zich in deze materie te verdiepen kan je vinden op Asynchronous Programming Overview. Deze link is wellicht niet beter dan uw eigen favoriete instap link.

4.7 Code plumbing, aka 'the strategy pattern'

Deze laatste topic behandelt het 'at run time' vervangen van code in *één welbepaald* object. Dit is helemaal anders dan overerving: bij overerving zullen we code at compile time aanpassen voor *elke* instantie van de ervende klasse.

4.7.1 Codeplumbing: een XMLViewer

Veronderstel dat u betrokken bent bij de ontwikkeling van een algemene XML-viewer. Deze is in staat om elke mogelijke XML-file in een treeview te visualiseren. Omdat u op voorhand niet alle XML-tags van uw potentële klanten kent kan u in uw XMLviewer geen tag- afhankelijke manipulaties coderen: u weet niet op voorhand hoe een XML-file met CD-gegevens vertoont moet worden. Het is onze doelstelling een XMLViewer control te ontwikkelen waarin de gebruiker bepaalde routines kan vervangen door zijn eigen implementaties. Wij voorzien default implementaties die een basis functionaliteit aanbieden zodat de XML-viewer op zijn minst een standaard gedrag vertoont. We gaan hiervoor als volgt te werk:

- maak een XMLViewer usercontrol die erft van TreeView. Deze overloopt bij het verwerken van een XML-file recursief de verschillende tags en toont deze. Wellicht is het interessant om ook een eigen TreeViewItem klasse te maken. Houd rekening met volgende elementen:
  - plaats de berekening van de getoonde tekst voor elke XML-node in een aparte routine (met de XML-node als argument);
  - plaats de berekening van de kleur voor elke XML-node in een aparte routine (met de XML-node als argument);
  - plaats de test die bepaalt of subnodes moeten verwerkt worden in een aparte routine (met de XML-node als argument);

  U merkt dat elke functionaliteit die we later voor de gebruiker aanpasbaar willen stellen in aparte routines terechtkomen. Deze routines implementeren het default gedrag (en mogen enkel gebruik maken van de parameters, niet van de module variabelen);
- voorzie voor elke default routine een delegate- class waar deze toe behoort;
- definieer voor elke default routine een field van het overeenkomstige delegate type, en koppel dit field in de constructor aan zijn default implementatie;
- vervang elke oproep van de default-routines door een *invoke*- aanroep van het field. Dit start de methode in het field waarin default onze eigen routines te vinden zijn.
- voorzie *setters* voor elk delegate- field in de klasse. Dit geeft gebruikers van onze XMLViewer de kans om onze default methodes door zijn eigen routines te vervangen.

Figuur 4.13: XMLViewer en code plumbing

De linker versie toont het default gedrag van de XMLViewer. De rechter versie toont dezeldfe informatie in dezelfde uitvoering nadat de XMLViewer control werd voorzien van aangepaste visualisatie routines.

De screenshot toont een WPF- programma. De techniek op zich is uiteraard ook bruikbaar in een WinForms context.

Jef Daels

4.7.2 Codeplumbing pitfalls

Wanneer we codeplumbing toepassen verdienen volgende punten uw aandacht:

- vermijd delegate properties indien u usercontrols maakt. Een property wordt door Visual Studio in het property window getoond (door de property te serialiseren). In het geval van een delegate zal dit in Visual Studio voor problemen zorgen (herhalende foutboodschappen, ook na het afsluiten en heropenen van het project in Visual Studio).
- laat uw codeplumbing routines *enkel* gebruik maken van de routine parameters. De delegate wordt immers uitgevoerd waar hij werd gedefinieerd. De module niveau variabelen zijn dus soms wel (wanneer de default routines worden uitgevoerd), soms niet (wanneer de nieuwe variant wordt uitgevoerd) deze van het object dat de delegate oproept.

4.8 Delegates: faits divers

4.8.1 Action en Func

Omdat in veel situaties gelijkaardige delegate definities nodig zijn werden in .NET een aantal delegate definities voorzien. Het is aan te raden deze indien mogelijk te gebruiken, eerder dan zelf nieuwe delegate klassen te definiëren;

4.8.1.1 Action

De klasse Action is een delegate klass voor routines zonder argumenten en zonder resultaat. Action$< T >$ is een (generische) klasse voor routines met exact één argument (gelijk welk type) en zonder resultaat.

4.8.1.2 Func

De generische klasse *Func*$< outTResult >$ wordt gebruikt voor functies zonder argumenten (en gelijk welk resultaat type), en *Func*$< inT, outTResult >$ wordt gebruikt voor functies met exact één argument (willekeurig type) en een willekeurig resultaat type.

4.8.2 Meer dan (begin)-invoke

Delegates ondersteunen meer dan enkel een (begin)invoke routine. Zo bestaat ook de mogelijkheid om verschillende delegates (van hetzelfde delegate type) te combineren (geen co- of contravariance mogelijk) en betaan er ook multicast delegates (die achter de schermen gebruikt worden om meerdere clients toe te laten op hetzelfde event te abonneren)

4.8.3 Timers

.NET ondersteunt verschillende Timer- classes (zie ook Comparing the Timer Classes in the .NET Framework Class Library). Indien u deze tekst doorneemt zal u begrijpen dat niet elke Timer 'tikt' op de UI-thread, wat betekent dat niet elke Timer geschikt is om UI-wijzigingen rechtstreeks aan te sturen (zie ook Asynchroon, met AsyncCallback, zonder UI).

Bemerk trouwens dat de constructor van de *System.Threading.Timer* klasse onder andere een delegate object verwacht. De twee andere timer-objecten werken met events, wat gezien Delegates en events eigenlijk niet echt een verschil uitmaakt.

4.8.4 Asynchrone eventverwerking

Wie de voorgaande topics goed begrepen heeft zou in staat moeten zijn om een eigen, asynchrone eventimplementatie op te zetten: een uitstekende oefening voor thuis.

4.8.5 Anonymous methods (.NET 2.0)

Bij het maken van een delegate instantie werd in voorgaande tekst telkens gebruikt gemaakt van een methode met gepaste signatuur. Indien men deze extra methode definitie wenst te vermijden (bijvoorbeeld bij het definiëren van MVVMCommands) kan men gebruik maken van *anonieme methodes*. Deze bestaan uit een (geschikte) formele parameterlijst definitie en een body:

```
btnAnonymousMethod.Click +=
    delegate(object o, RoutedEventArgs e)
        {MessageBox.Show("test1"); MessageBox.Show("test2");}
    ;
```

Figuur 4.14: Anonymous method voorbeeld

4.8.6 Lambda expressions (.NET 3.0)

Lambda expressions kunnen onder andere gebruikt worden om delegate objecten te creeëren. Ze bestaan uit twee stukken gescheiden door middel van de lambda operator =>. Links bevindt zich de input (het equivalent van de formele parameterlijst van een functiehoofding), rechts bevindt zich de body van de delegate. Het resultaat van de lambda expressie is (in dit geval) een delegate object:

```
btnAnonymousMethod.Click +=
    (object o, RoutedEventArgs e) =>
        {MessageBox.Show("test3"); MessageBox.Show("test4");
    ;
```

Figuur 4.15: Lambda expressie voorbeeld

Lambda expressions worden veel gebruikt in een LINQ-context, waar het injecteren van bijvoorbeeld selectielogica in een foreach-lus frequent voorkomt.

4.9 Delegates: niet voor elk programma een geschikte oplossing

- niet elk programma heeft nood aan delegates;
- (algemener:) niet elke 3nMCT-techniek is noodzakelijk om een goed programma te maken;
- (advies:) indien uw programma geen nood heeft aan delegates, gebruik ze dan niet: er zijn wellicht meer programma's te ontwikkelen zonder delegate- nood dan met.

4.10 Delegates: opgaves

- events: maak een eigen TextBox klasse met een extra event (guidelines!): dit event wordt afgevuurd indien het tekstvak leeg wordt. Demonstreer uw oplossing.
- multithreading: maak een window waarin verschillende tellers asynchroon kunnen oplopen. De teller wordt gerealiseerd door een control die erft van TextBox. Om dit te realiseren zal u de prioriteit van de visualisatie op background priority moeten uitvoeren. De demo toont de volgende elementen:
  - singlethreaded is er maar één activiteit actief;
  - multithreaded zijn er verschillende acties terzelfdertijd mogelijk: het tellen in twee textboxen en het verplaatsen van het window.
- maak een XMLViewer waarin de klant eigen routines kan inpluggen om specifieke xml's anders te visualiseren (zie ook XMLViewer). Demonstreer uw oplossing.

Jef Daels

Hoofdstuk 5

Reflection

5.1 Reflection: duiding

Reflection is technologie waardoor (meta-) informatie omtrent programma-elementen programmatorisch toegankelijk wordt voor eigen programma's. Deze programma-elementen bevinden zich veelal in *assemblies* die niet tot het uitvoerende programma zelf behoren (externe modules). Op basis van de bekomen informatie zal ons eigen programma wellicht extra functionaliteit aanbieden. In dit hoofdstuk zullen we reflection gebruiken om een *plugin* infrastructuur op te zetten.

Hoewel reflection, net als delegates, de mogelijkheid biedt om routines te starten zou het verkeerd zijn om delegates als onderdeel van reflection te interpreteren (of omgekeerd). Een belangrijk verschil is mijns insziens de op voorhand gekende interface elementen van een delegate- instantie (*invoke*, ..) terwijl reflection net de mogelijkheid biedt om met at-compile-time ongekende klasses en/of routines te werken.

Het belang van reflection kan moeilijk overschat worden. Visual Studio illustreert mooi de mogelijkheden van reflection: intellisense, het property-window, de compiler .. : veel van de programmeur ondersteunende features maken gebruik van reflection, en in het bijzonder van attributen. Ook het hoofdstuk omtrent Aspect Oriented Programming steunt sterk op reflection.

Tot slot zal de in dit hoofdstuk opgedane kennis toelaten om object oriëntatie als een syntactische constructie bovenop procedureel programmeren te zien: een inzicht dat hopelijk de blik verruimt.

5.1.1 Relevante namespaces

- *System.ComponentModel*
- *System.Reflection*

5.2 Reflection: attributen voorbeeld

Een programmeur kan code elementen voorzien van attributen om extra informatie aan het code element te koppelen. Deze attributen kunnen door andere programma's gelezen worden om een gepaste verwerking van het element mogelijk te maken. Deze verwerking staat los van de werking van het code element zelf die wellicht niet gewijzigd wordt. De namespace *System.ComponentModel* bevat attributen die door Visual Studio gebruikt worden. Een aantal van deze zijn enkel van toepassing in een WinForms context, en niet in WPF:

Figuur 5.1: ComponentModel attributen ($C^{\#}$)

Voor bovenstaande voorbeeld kunnen we volgende elementen vermelden:

- *DefaulEvent* en *DefaultProperty* worden enkel toegepast op klasse (usercontrol) niveau. Ze worden door Visual Studio gebruikt om in de designer volgende acties te ondernemen:
  - *DefaultProperty*: deze property wordt in het property window automatisch geslecteerd wanneer een control van deze klasse in de (Visual Studio-) designer wordt geselecteerd;
  - *DefaultEvent*: dit event wordt automatisch geselecteerd in het property window (event tab) wanneer een control van deze klasse in de (Visual Studio) designer wordt geselecteerd. Dubbel klikken op deze control zal code genereren om dit event op te vangen.
- *Category* is een attribuut waarmee de category (gebruikt in het propertywindow) van een property wordt opgegeven. Deze wordt door Visual Studio gebruikt (zie figuur);
- *Description* en *DefaultValue* zijn attributen die enkel in een Winforms Visual Studio designer omgeving gebruikt worden (de WPF designer in Visual Studio negeert deze attributen).

5.3 Reflection in .NET: meer dan attributen

5.3.1 Reflection in .NET: start

Maak een nieuwe solution met hierin een WPF-project. Maak een tweede(!) WPF project met volgende basis functionaliteit:

- voorzie een klasse *clsPersoon* met een aantal features;
- voorzie een knop op het default window om een nieuwe persoon aan te maken.

Compileer het nieuwe project en programmeer onderstaande code in het eerste project (zonder een referentie tussen beide projecten te leggen):

```
private void verwerkDemoUI()
    {
    Microsoft.Win32.OpenFileDialog ofd = new Microsoft.Win32.OpenFileDialog();
    ofd.Filter = ".NET assemblies|*.exe;*.dll";
    ofd.InitialDirectory = System.IO.Path.GetDirectoryName(
        System.Reflection.Assembly.GetExecutingAssembly().CodeBase);

    if (ofd.ShowDialog().Value)
        verwerkDemoUI(ofd.FileName);
    }
private void verwerkDemoUI(string sAssembly)
    {
    Assembly ass = Assembly.LoadFile(sAssembly);
    foreach (Type t in ass.GetTypes())
        Console.WriteLine(t.Name);
    }
```

Figuur 5.2: Assembly reflection ($C^{\#}$)

Indien we bij uitvoering van deze code en het tonen van de *OpenFileDialog* het compilatie resultaat van het tweede project selecteren, zal het uitvoerende programma in de console de naam van elk type in de gekozen assembly tonen: *MainWindow*, *App*, *Resources*, *Settings* en *clsPersoon*. Eenvoudig nazicht in Visual Studio leert dat dit klopt.

Bovenstaande code is een reflection startpunt: vertrekkend vanuit een assembly wordt informatie omtrent de code in de assembly opgevraagd (bijvoorbeeld de namen van de klasses). De Type-klasse ondersteunt flink wat getters: zo kunnen de attributen, constructoren, methodes, properties, events, ... worden opgevraagd. Elk van deze code elementen is terug ondervraagbaar. Het wordt sterk aangeraden om een aantal getters zelf te verwerken, ook al vind u hiervan geen afbeeldingen in deze tekst terug.

5.3.2 Hoe een instantie maken van een klasse in een assembly?

Normaal gezien zullen we, voor een gekende klasse clsX, volgende $C^{\#}$- syntax gebruiken:

```
clsX x = new clsX(..);
```

Figuur 5.3: Strongly typed instantiatie

In bovenstaande code wordt de gekende klasse clsX op twee plaatsen gebruikt: als type van de referentie en als type waarvan de constructor wordt opgeroepen. Indien we een type bekomen via reflection is bovenstaande onmogelijk: gezien het nodige type at compile time niet gekend is (clsX wordt gevonden in een at run-time geladen assembly) is bovenstaande lijn code onmogelijk (in $C^{\#}$), en moeten we twee problemen oplossen:

- het referentie type zullen we vervangen door *System.Object*. Omdat elke klasse (al dan niet indirect) erft van *System.Object* is elk instantie van een klasse te koppelen aan een referentie van dit root- type (polymorfisme door overerving). Natuurlijk kunnen we, indien we extra kennis omtrent de klasses hebben, eventueel een meer gespecialiseerd type gebruiken (zie het voorbeeld hieronder);
- het oproepen van de constructor van een klasse zal gecodeerd worden via de *Activator*-klasse:

```
private void verwerkIndienWindow(Type t)
    {
    if (t.Name.Equals("MainWindow"))    //dergelijke test vermijden => attributes
        {
        Window wdw = (Window) Activator.CreateInstance(t);
        wdw.Show();
        }
    }
```

Figuur 5.4: Activator.CreateInstance ($C^{\#}$)

- indien de constructor argumenten vereist kunnen deze aan de *CreateInstance*- oproep worden meegegeven (u bent *niet* beperkt tot parameterloze constructoren);
- omdat we weten dat een klasse met deze naam (beslissingen op basis van een klasse naam zijn te vermijden!) een Window is gebruiken we een *Window* referentie waarvan de *Show*-method *at compile time* gekend is.

Een iets gecompliceerder voorbeeld illustreert het starten van een methode vertrekkend vanuit zijn object referentie:

```
private void verwerkIndienPersoon(Type t)
    {
    if (t.Name.Equals("clsPersoon"))  //dergelijke test vermijden => attributen
        {
        object p = Activator.CreateInstance(t);
        t.InvokeMember("FNaam", BindingFlags.SetProperty, null, p, new object[] { "Familie" });
        t.InvokeMember("VNaam", BindingFlags.SetProperty, null, p, new object[] { "lid" });

        string sPersoon = (string) t.InvokeMember("ToString", BindingFlags.InvokeMethod, null, p, new object[] { });
        Console.WriteLine(sPersoon);
        }
```

Figuur 5.5: Activator.InvokeMember ($C^{\#}$)

- de Type- klasse ondersteunt een *InvokeMember* methode waarmee een methode kan gestart worden. Het object dat de methode uitvoert is een argument van deze oproep, de nodige argumenten worden als een array van objecten voorzien;
- deze manier van werken is case sensitief!
- deze voorbeeld code is weinig representatief: de kans is klein dat de klasse- naam op voorhand gekend is, net zomin als men mag verwachten de methodes op voorhand te kennen. Indien de methodes op voorhand gekend zijn is het wellicht aangewezen om met een interface te werken. Op die manier is een verwerking als deze van het window (zie vorige voorbeeld) mogelijk.

5.3.3 .NET en attributes

5.3.3.1 .NET en attribute informatie opvragen

De klasse *Type* ondersteunt een *GetCustomAttributes* methode waarmee de (custom-) attributen van een klasse kunnen opgevraagd worden. Ook andere code elementen hebben een *GetCustomAttributes* property. Op basis van de extra informatie gevonden in een attribuut kan een gebruikers programma gestuurd worden. Verschillende attribute klasses ondersteunen natuurlijk verschillende properties. Een voorbeeld attribuut zullen we uitwerken in onze eigen plugin infrastructuur implementatie.

5.3.3.2 Hoe zelf Attribute klasses maken in .NET?

Een goede beschrijving kan u vinden op Writing Custom Attributes. Als opmerking hierbij zou ik formuleren dat ik verwacht dat de meeste properties *read-only* gedefinieerd zijn (en niet *read-write* zoals in de voorbeeldcode).

Er wordt zeker verwacht dat u in staat bent om attributen te definiëren die enkel op classes of enkel op methodes toegepast kunnen worden.

5.4 Een eigen plugin- infrastructuur!

5.4.1 Een eigen plugin- infrastructuur: vereisten

Ik wens volgende situatie te realiseren:

- we moeten de mogelijkheid voorzien om in een toepassing extra logica uit een op voorhand niet gekende *assembly* te laden;
- de extra logica wordt aangeboden door middel van plugins. Een plugin- klasse erft van *Window*, en alle relevante extra's worden ontsloten via deze klasses. Er kunnen Window klasses in de assembly worden gevonden die geen plugins zijn (maar bijvoorbeeld gestart kunnen worden door een plugin in dezelfde assembly);
- het is de bedoeling om bij het laden van een assembly alle gevonden plugins via het menu te presenteren. De menu-tekst voor een plugin moet ook in de assembly gevonden worden;
- zie demo.

5.4.2 Een eigen plugin- infrastructuur: mogelijke oplossing

Voorgaande vereisten kunnen op verschillende wijzes gerealiseerd worden. De hier geformuleerde oplossing is attribute- gebaseerd:

- het onderscheid maken (differentiëren) tussen klasses die al dan niet als plugin moeten aanzien worden kan door middel van een attribuut. Dit attribuut ondersteunt twee properties: het al dan niet plugin zijn (isPlugin, bool), en de omschrijving van de eventuele plugin (description, String). Dit resulteert in een plugin- klasse met twee fields, twee (read only) properties en een constructor met twee argumenten;
  - indien een klasse dit attribuut niet heeft is de klasse geen plugin (default).
- indien een klasse het plugin attribuut heeft, en isPlugin is true, dan is dit een plugin klasse waarvoor we een extra menu item zullen voorzien (in de software die de plugin-assembly verwerkt). Indien op dit menu item wordt geklikt wordt een instantie van de plugin klasse aangemaakt en getoond
  - we veronderstellen (leggen op) dat het plugin attribuut enkel is toegepast geworden op Window- klasses;
  - het is *niet* de bedoeling om in het client- window een boekhouding tussen menuitems en plugin- klasses bij te houden: het menu-item zelf moet bij het klikken alle nodige bewerkingen onafhankelijk kunnen afhandelen.
    - dit laatste punt is belangrijk en lijkt soms in de totaal oefening verloren te gaan.

5.4.3 Een eigen plugin- infrastructuur: implementatie

Zowel de plugin- assembly als het client- programma (het programma dat de plugins wil laden) moeten toegang hebben tot dezelfde plugin- attribuut klasse. Omdat er geen referentie tussen de plugin- en de client- assembly mag liggen (anders is de plugin assembly at compile time gekend en is de ganse plugin- opzet overbodig) en beide assemblies toch dezelfde attribuutklasse moeten kennen zijn we verplicht het plugin- attribuut in een apart project te plaatsen (waar zowel het plugin- project als het client- project naar refereren). We hebben dus minimum drie projecten! Omtrent die drie projecten kunnen we volgende vragen stellen:

- welk project type is elk project?
- welke referenties liggen tussen deze projecten?
- waarom hebben we eigenlijk drie projecten nodig? Indien het plugin- en client project beide dezelfde bron-file voor de attribuut klasse bevatten, is dit niet voldoende?
- formuleer een oplossing met maar twee projecten (perfect mogelijk trouwens, minder goed, naam gebaseerd).

Maak in dit derde project een attribuut klasse met volgende eigenschappen:

- volg de naamgeving guideline!
- voorzie de mogelijkheid om het plugin zijn en een plugin- omschrijving te bewaren, toe te kennen via de constructor en op te vragen via read only properties;
- maak het onmogelijk om dit attribuut toe te passen op andere code elementen dan klasse definities.

Maak in het tweede project (het plugin- project dat plugins aanlevert) een window aan en koppel wat logica aan het klikken op een knop. Plaats het plugin- attribuut op de MainWindow klasse en niet op de klasse clsPersoon.

Het clientproject dat de eventuele plugins zal laden gaat als volgt tewerk:

- bij het opstarten van een window dat de plugins ter beschikking stelt (de constructor van dit window is een goede plaats) lezen we alle assembly files gevonden op een vooraf afgesproken plaats (in mijn eigen oplossing is dit de plugins subdirectory van de toepassing)
  - System.IO is uw favoriete namespace voor dergelijke zaken;
  - gebruik een filter om enkel de assembly files op te vragen;
  - in een echte toepassing zou de plugin-directory configureerbaar moeten zijn.
- de types in elke *dll*- assembly worden overlopen. Indien ze voorzien zijn van een plugin attribuut (en isPlugin is *true*) voorzien we een extra menu item in het (main-)menu van dit window. Dit nieuwe menu-item is een instantie van een eigen menuitem klasse:

  - ze erft van de klasse *MenuItem*;
  - de omschrijving getoond in het menu is de description gevonden in het plugin attribuut;
  - ze implementeert een eventhandler zodat bij het klikken op een menu-item van deze klasse een istantie van het geassocieerde plugin-type wordt gemaakt en getoond;
  - bepaal zelf de nodige constructor argumenten.

5.4.4 Eigen plugin infrastructuur: demo

Gedemonstreerde elementen:

- het verwijderen van plugin assemblies verwijdert de plugins uit het menu;
- indien *isPlugin false* terug geeft wordt de eventuele plugin niet getoond;
- het verwijderen van het plugin attribuut zal (na hercompilatie) de plugin niet meer tonen in het menu;
- het plaatsen van het plugin- attribuut op een niet window klasse resulteert uiteraard in een foute werking. Een goed programma zal hierop niet crashen!
- er wordt in het window met de menu-items geen koppeling tussen de menu items en het plugin type voorzien: de volledige afhandeling om een plugin window te starten dient in de eigen plugin menu item-klasse geprogrammeerd te zijn.

5.5 Namespaces met interessante attributen

- System.Diagnostics: vooral debug en performance attributes;
  - ConditionalAttribute: zeer interessant wanneer u in uw code het onderscheid wensts te maken tussen Debug- en Release- uitvoeringen (klik nu op de link :)).
- System.Componentmodel: vooral component en control gerichte klasses (zie msdn);
  - onder andere tal van convertoren (niet echt IDE stuff);
  - licensing attributen (!);

5.6 Reflection: opgaves

- ontwerp een plugin infrastructuur (analoog met de demo);
- pas het fractalen programma aan zodat het tonen van de drietallige boom en Sierpinski als plugin in een programma wordt opgenomen.

Hoofdstuk 6

Runtime compilatie

6.1 Runtime compilatie: duiding

De meeste softwareontwikkelingen bestaan uit twee stappen: er wordt code geschreven (of gegenereerd) (stap één), waarna deze gecompileerd wordt tot een uitvoerbaar programma (stap twee). Eventueel zal een toepassing verschillende projecten bevatten die elk gecodeerd en gecompileerd worden. Dit beperkt uiteraard de mogelijkheden van de toepassing: wat niet op voorhand werd geprogrammeerd (eventueel in een PlugIn) kan niet uitgevoerd worden. Indien we een tekenprogramma wensen te ontwikkelen, waarbij de gebruiker het functievoorschrift intikt hebben we een probleem (we wensen niet zelf een wiskunde parser te schrijven). Ook indien we een toepassing maken waarbij de beveiliging van sommige onderdelen door de klant wordt bepaald kunnen we niet altijd alle mogelijkheden op voorhand voorzien. Sommige database toepassingen zullen een property list in een tabel bewaren (bv.: de talen waarin een produkt gekend moet zijn, zie ook de vertalingen demo): ook hier is het niet mogelijk om op voorhand alle properties in een klasse te compileren. In dergelijke situaties is het interessant om, gedurende de uitvoering van het programma, een extra stukje code te genereren, compileren en uitvoeren: *runtime compilatie* (ook wel on the fly compilatie geheten), zodat databinding zijn rol kan spelen.

Recentelijk werden de .NET compilers herschreven (Roslyn) zodat u indien u dat wenst de compiler informatie kan bekomen en wijzigen. Hoewel dit allerhande code manipulerende mogelijkheden biedt heb ik er nog geen gevonden waar ik het wenselijk acht om een eigen implementatie uit te werken.

6.1.1 Relevante namespaces

- *System.CodeDom.Compiler*

6.2 .NET compilaties starten

Zowel de $C^{\#}$als de VB.NET compilers maken deel uit van de CLR. Afhankelijk van de gegenereerde code ($C^{\#}$of VB.NET) zullen we dan ook de juiste compiler moeten oproepen. We zullen de code en het compilatieresultaat (de assembly) in memory houden zodat er geen files overblijven nadat ons programma eindigt.

6.2.1 Command line compilatie

Wellicht compileerde u tot nu toe altijd met behulp van Visual Studio. Het achterliggende compilatie proces (in welke volgorde compileren, welke compilatie parameters zijn nodig) wordt volledig verborgen (een voordeel). Om het compilatie proces duidelijker te maken zullen we in een eerste stap een volledig (console-) programma editeren, compileren en uitvoeren zonder gebruik te maken van Visual Studio.

6.2.1.1 Command line compilatie: $C^{\#}$

- maak een testfile abc.cs met een ascii- tekstverwerker (uw code files mogen enkel programma code bevatten, een tekstverwerker als Word is dus niet geschikt) en codeer volgende elementen ($C^{\#}$is case sensitive!):
  - voorzie een gepaste Main- methode (zie ook programmeren II: elk programma start door het uitvoeren van een static Main methode; of lees Main() and Command Line Arguments ($C^{\#}$Programming Guide)). Uiteraard moet u deze methode in een klasse coderen. Anders dan in JAVA moet de klasse naam niet identiek zijn aan de filenaam;
    - de klasse *String* zal enkel gekend zijn indien u ze prefixt met de namespace *System* of wanneer u een overeenkomstig *using* statement codeert;
  - schrijf het eerste runtime argument weg in de Console;

```
using System;
public class Test
        {
        public static void Main(String[] args)
                {
                Console.WriteLine(args[0]);
                }
        }
```

Figuur 6.1: Commandline compilatie: $C^{\#}$broncode

- start een Visual Studio command prompt (
  - win8: tik op het startscherm *tools* en kies de *native tools command prompt*;
  - win7: Start.All Programs.Microsoft Visual Studio xxx.Visual Studio Tools.Visual Studio Command prompt(xxx);
  - csc.exe is de $C^{\#}$compiler. U dient deze op te roepen met de juiste set parameters (*csc /?* toont u alle mogelijke opties met een summiere toelichting). Voor ons eenvoudige programma is de */out:* parameter eigenlijk overbodig (indien geen out- parameter wordt opgegeven heeft de resulterende assembly dezelfde naam als de bron-file);
  - met de target- parameter (niet in het voorbeeld) kan u aangeven of u een .exe (default) of .dll assembly wenst te bekomen;

Figuur 6.2: $C^{\#}$commandline compilatie en uitvoering

- indien ons programma gebruik zou maken van niet standaard bibliotheken moeten deze bij compilatie uiteraard ook worden meegegeven. Voor een gewoon WinForms of WPF programma wordt de parameterlijst zo groot (ongeveer elke project property komt terug als compiler optie), en is de compilatie volgorde zo belangrijk dat we dit zeker niet manueel wensen uit te voeren. Gedurende het programmeren zal Visual Studio dit voor ons doen. Indien u het proces wenst te automatiseren (om een groot project elke nacht volledig te hercompileren) kan u zich best wat verdiepen in MSBuild.

6.2.1.2 Command line compilatie: VB.NET

We volgen gelijkaardige stappen, maar dan voor VB.NET:

- VB.NET code:

```
imports System

public class test
   public shared sub main(args as String())
         Console.writeline(args(0))
   end sub
end class
```

Figuur 6.3: Commandline compilatie: VB.NET broncode

- VB.NET compilatie en uitvoering:

Figuur 6.4: VB.NET commandline compilatie voorbeeld

Het coderen en compileren is voor beide programmeertalen sterk gelijkaardig.

6.2.1.3 Een extraatje omtrent MSBuild

Het compileer proces van MSBuild gebeurt door middel van informatie gevonden in tekst files (zie ook MSBuild .Targets Files). Indien u hierin onderlegt bent kan u deze aanpassen (neen, u bent het wellicht niet). Aanpassen van deze files is de manier waarop PostSharp er in slaagt om het compilatie proces aan te passen zonder dat u als programmeur dient tussen te komen.

Jef Desmedt

6.2.2 Runtime compilatie: System.CodeDom.Compiler

Het is natuurlijk niet de bedoeling om voor normale sitaties een eigen compilatie proces uit te werken. Het is wel onze bedoeling om, vanuit een programma, programmacode te genereren en compileren (tot een .dll of .exe), en de bekomen assembly te verwerken zoals reeds werd gezien in reflection.

De .NET compilers zijn beschikbaar via de *System.CodeDom-* namespace:

System.CodeDom.Compiler Namespace

The **System.CodeDom.Compiler** namespace contains types for managing the generation and compilation of source code in supported programming languages. Code generators can each produce source code in a particular programming language based on the structure of Code Document Object Model (CodeDOM) source code models consisting of elements provided by the System.CodeDom namespace.

Classes

Class	Description
CodeCompiler	Provides an example implementation of the ICodeCompiler interface.
CodeDomProvider	Provides a base class for CodeDomProvider implementations. This class is abstract.
CodeGenerator	Provides an example implementation of the ICodeGenerator interface. This class is abstract.
CodeGeneratorOptions	Represents a set of options used by a code generator.
CodeParser	Provides an empty implementation of the ICodeParser interface.
CompilerError	Represents a compiler error or warning.
CompilerErrorCollection	Represents a collection of CompilerError objects.
CompilerInfo	Represents the configuration settings of a language provider. This class cannot be inherited.
CompilerParameters	Represents the parameters used to invoke a compiler.
CompilerResults	Represents the results of compilation that are returned from a compiler.
Executor	Provides command execution functions for invoking compilers. This class cannot be inherited.
GeneratedCodeAttribute	Identifies code generated by a tool. This class cannot be inherited.
IndentedTextWriter	Provides a text writer that can indent new lines by a tab string token.
TempFileCollection	Represents a collection of temporary files.

Figuur 6.5: *C#*System.CodeDom.Compiler namespace

Bemerk dat er geen aparte klasses per programmeertaal zijn:

- de klasse *CodeDomProvider* geeft toegang tot de .NET- compilers. De static methode *CreateProvider* heeft een argument waarmee wordt aangegeven voor welke programmeertaal we een compiler wensen te bekomen;
  - indien u twijfelt omtrent de taal-parametertekst (c# of csharp? VB of VB.NET? case sensitive?) kan u deze bekomen door de static methode *GetLanguageFromExtension* op te roepen (de file extentie moet u dan uiteraard wel kennen);
- *CompilerParameters* is handiger dan de commandline benadering: de verschillende parameters zijn niet als een lange tekst te formuleren maar zijn properties op een CompilerParameters object;
  - het toevoegen van een assembly kan gebeuren door middel van zijn naam: *System.Data.dll*
  - een assembly waarin een type XYZ zich bevind kan worden teruggevonden door *typeof(XYZ).Assembly.Location*

- *CompilerResults* bevat het compilatie resultaat. Dit bevat zowel de compilatie feedback (error, warning, ..) als de (eventueel) resulterende assembly;

6.3 CodeDom

De eerder aan bod gekomen oplossing verwacht broncode die gecompileerd kan worden: deze is natuurlijk programmeertaal afhankelijk. In een andere benadering kunnen we een programmeer boom maken waarin programmeer- entiteiten worden gehangen (types, methodes, variabelen, commando's, ..) waarvoor achteraf programma code (in de programmeertaal naar keuze) wordt gegenereerd:

```
CodeCompileUnit ccunit = new CodeCompileUnit();
CodeNamespace cns = new CodeNamespace("taalprog");
ccunit.Namespaces.Add(cns);

CodeTypeDeclaration klas = new CodeTypeDeclaration("klasnaam");
cns.Types.Add(klas);
klas.Attributes = MemberAttributes.Public;
```

Figuur 6.6: CodeDOM

We spreken in deze context over een abstract syntax tree.

6.4 System.Reflection.emit

Voor de volledigheid vermelden we ook de System.Reflection.Emit Namespace. Deze laat toe om bij uitvoering *IL-* code genereren (normaal gezien het resultaat van een compilatie). Dit valt jammer genoeg buiten het bereik van deze cursus en mijn kennis.

6.5 Het Roslyn project

Een interessante ontwikkeling in het compilatie gebeuren van .NET is de ontwikkeling van het Roslyn project (momenteel nog in CTP (community technical preview)). Hierbij worden tussentijdse resultaten (de abstract syntax tree, koppeling tussen namen en objecten, ..) van het compilatie proces ontsloten naar ontwikkelaars toe. Dit biedt gigantisch veel mogelijkheden naar geautomatiseerde code processing (zeker naar rapportering toe, wellicht ook naar het automatisch aanpassen van code toe).

6.6 Assembly code inspecteren

Het resultaat van een .NET compilatie is (indien gelukt) een assembly. Deze bestaat uit *IL*(Intermediate language) code: de machinetaal van .NET. Deze assemblies kunnen door middel van disassemblers gemakkelijk gedecompileerd worden. Dit laat derden toe om uw broncode grotendeels te reproduceren wat meestal niet wenselijk is. Het gebruik van obfuscators kan de leesbaarheid van de gedecompileerde broncode sterk omlaag halen.

Deze problematiek is niet .NET specifiek: ook byte code (het java equivalent van IL- code) kan gedecompileerd worden. Ook voor C(++) output zijn er disassemblers te vinden. In de .NET context zal u merken dat u een assembly kan decompileren naar uw programmeertaal naar keus, wat ook de oorspronkelijke programmeertaal was.

6.6.1 ILDasm

ILDasm is de *IL* disassembler meegeleverd met *Visual Studio*. Deze kan gebruikt worden om *IL-* code te tonen in een textfile formaat. Op deze manier is het mogelijk om bijvoorbeeld de aanwezigheid van attributen na te kijken. Hoewel *ILDasm* zeker zijn plaats heeft (zie ook Ildasm.exe (MSIL Disassembler)) is het zeker niet de meest geschikte tool om de brontekst van een programma na te kijken:

Figuur 6.7: ILDasm output

6.6.2 ILSpy

ILSpy is het gratis antwoord waarmee werd gestart nadat de voorheen gratis Reflector software betalend werd. Voor zover ik beide oppervlakkig gebruikt heb vind ik ze sterk gelijkwaardig: beide laten toe om een door ons gecompileerd programma om te vormen naar een .NET-taal naar keuze.

Bij het downloaden van ILSpy kan u kiezen tussen de broncode en de gecompileerde bestanden. Ikzelf verkies meestal de broncode, maar het staat u uiteraard vrij om de binaries te installeren.

Gelieve na het installeren *ILSpy* eens op te starten en een programma te decompileren:

Figuur 6.8: ILSpy output

Bemerk dat in bovenstaande figuur VB.NET- code werd gegenereerd terwijl de oorspronkelijke programmatekst in *C#*werd geschreven.

6.6.3 Reflector

Deze (niet langer gratis) tool is (niet langer) wat u zoekt! Hiermee bent u in staat om een assembly te decompileren in *de programmeertaal naar keus*. Indien u de nodige plugins installeert kan u een assembly volledig decompileren tot een project (sommige namen zullen wel niet compileerbaar zijn, maar dat is voor kwaadwillige derden slechts een kleine beperking):

Figuur 6.9: Reflector output

Bemerk dat in bovenstaande figuur VB.NET- code werd gegenereerd terwijl de oorspronkelijke programmatekst in $C^{\#}$werd geschreven. In de programma code zijn de groene tekst elementen hyperlinks (niet beschikbaar in ILSpy): hiermee kan u navigeren naar de definitie van dit element (ongeveer zoals *F12* in Visual Studio).

6.6.4 Obfuscatie

Obfuscatie is een techniek om een assembly aan te maken die ook na decompilatie quasi onleesbaar is. De (gratis) versie die geïnstalleerd wordt met Visual Studio (onder tools) vertrekt van een compilatie resultaat en zal alle hierin gevonden namen (klasses, methodes, parameters, ..) *scramblen*, wat het geheel voor een complex programma quasi onleesbaar maakt. Het is uiteraard niet wenselijk om de broncode van het project zelf te scramblen, omdat u in dat geval geen aanpassingen meer zou kunnen doorvoeren.

Figuur 6.10: Obfuscated output via Reflector

Bemerk in bovenstaande figuur het hernoemd zijn van de controls. Veel routine namen zijn niet hernoemd geworden, wat spijtig is. Wellicht zal de betalende versie hier een betere obfuscatie mogelijk maken.

6.6.4.1 Confuser

Een betere obfuscater vindt u op Confuser. Het resultaat van deze obfuscater kan niet geopend worden in ILDASM en is totaal onleesbaar wanneer u bijvoorbeeld ILSpy gebruikt.

6.7 Opgaves

6.7.1 Een tekenprogramma

Het te realiseren programma moet aan volgende vereisten voldoen:

- de gebruiker kan zelf een functievoorschrift intikken (reden waarom we on-the-fly compilatie nodig hebben: we zullen dit functievoorschrift compileren zodat we het kunnen uitvoeren);
- het tekenen gebeurt in een usercontrol, waarbinnen de tekening geschaald wordt getoond: we rekken (of krimpen) de te tekenen figuur zodat de bovenkant van de figuur de bovenkant van de usercontrol raakt (idem voor de onderkant);

Figuur 6.11: Tekenen van een functie

6.7.2 ILSpy

Op het examen wordt verwacht dat u volgende manipulaties vlot kan uitvoeren:

- installeer ILSpy en pas deze toe op een assembly naar keus;
- demonstreer dat u kan decompileren naar verschillende programmeer talen.

6.7.3 Obfuscatie

Op het examen wordt verwacht dat u volgende manipulaties vlot kan uitvoeren:

- gebruik *Confuser* om een assembly naar keus te obfuscaten;
- gebruik de obfuscator (gevonden onder tools) om een assembly naar keus te obfuscaten;
- gebruik ILSpy om het obfuscated zijn van uw nieuwe assembly te demonstreren.

Jef Daels

Hoofdstuk 7

Backtracking

7.1 Backtracking: duiding

De eerste computers werden oorspronkelijk gebruikt om oplossingen te bieden voor problemen die mensen niet (snel genoeg) konden oplossen. Dit hoofdstuk behandelt het gebruik van een computer om problemen op te lossen waarvoor we zelf, zonder computer, geen antwoord kunnen formuleren. De gebruikte techniek heet *backtracking*: hij is niet op alle problemen toepasbaar, is soms hopeloos inefficient maar is een haalbare kaart om artificiële intelligentie te introduceren. In de laatste sectie zullen we backtracking gebruiken om AI in een spelletjes context te introduceren.

7.1.1 Relevante namespaces

7.1.2 Backtracking: principe

Backtracking is een benadering waarbij men, door het overlopen van alle mogelijke oplossingen, zoekt naar een beste (eerste, ..) oplossing. De implementatie van het algorimte is meestal recursief, met hierin volgende stappen:

- *stopconditie*: indien de oplossing werd gevonden geven we true terug;
- anders worden alle mogelijkheden voor deze stap in een lus overlopen:
  - voer één van de stap mogelijkheden uit;
  - indien de recursieve oproep voor de volgende stap slaagt hebben we een oplossing gevonden: we geven true terug;
  - indien de recursieve oproep faalt maken we de zopas gezette stap ongedaan en vervangen deze door één van de andere mogelijkheden, en herhalen de recursieve stap;
- indien geen enkele van de mogelijkheden tot een oplossing leidt moeten we concluderen dat de eerder gezette stappen het vinden van een oplossing blokkeren, en keren we terug naar de oproeper (return false), die dan op zijn beurt een volgende mogelijkheid kan onderzoeken.

7.2 N-Queens probleem

7.2.1 Probleemstelling, synchrone oplossing

Een gemakkelijk uit te leggen backtrack probleem is het N-Queens probleem: hoe plaats ik N koninginnen op een schaakbord zodat deze elkaar niet slaan. Twee koninginnen slaan elkaar indien ze:

- op dezelfde kolom staan;
- op dezelfde rij staan;
- uit de eerste koningin een diagonaal vertrekt waarop de tweede koningin zich bevindt.

Een geïllustreerde probleemstelling vind u op Eight queens puzzle. De toelichting van het algoritme gebeurt tijdens de les.

Figuur 7.1: NQueens oplossen

7.2.2 Asynchroon: BeginInvoke/4

Indien we grotere aantallen (bijvoorbeeld 30) koninginnen dienen te plaatsen neemt de berekening flink wat tijd in beslag. Om de interface hiermee niet te blokkeren zullen we de berekening van de oplossing op een andere thread laten uitvoeren. Hiertoe gaan we als volgt te werk:

- maak een delegate-class voor de zoekroutine;
- start een asynchrone oproep voor deze zoekroutine, en voorzie een callback waarmee het berekende resultaat in de UI kan getoond worden (u zal hoogst waarschijnlijk een state object nodig hebben);
- laat de asynchrone thread de berekende oplossing in de UI tonen (let op de UI-thread pitfall!).

Hint: om te vermijden dat gedurende de asynchrone zoekoperatie een volgende zoekoperatie gestart zou worden (er kan immers geklikt worden in de UI terwijl de zoekoperatie loopt) is het best om de UI-elementen die de zoekopdracht starten of specifiëren disabled worden geplaatst: het is immers niet wenselijk de zoek- informatie te wijzigen terwijl de vorige zoekopdracht nog loopt (de getoonde resultaten zouden niet langer overeenstemmen met de zichtbare criteria) net zomin als we een volgende zoekopdracht willen starten terwijl de vorige nog niet ten einde was. Als oefening kan eventueel een cancel- button voorzien worden.

7.2.3 Asynchroon: async/await

Hiervoor verwijs ik naar Await/Async verder in deze tekst. Deze is ook bruikbaar voor de Sudoku- toepassing die straks aan bod komt.

7.3 Sudoku's oplossen

Het oplossen van Sudoku's is niet altijd eenvoudig. Een computerprogramma kan u behulpzaam zijn, maar ontneemt u wellicht het liefhebbers plezier.

7.3.1 Sudoku: spelregels

Het invullen van een 9x9 sudoku gebeurt op volgende wijze:

- elk hokje van de matrix moet worden ingevuld met een geheel getal tussen 1 en 9 (grenzen inbegrepen);
- in een rij mogen geen dubbels voorkomen;
- in een kolom mogen geen dubbels voorkomen;
- in elk van de negen 3x3 hokjes (naast elkaar, niet overlappend) mogen er geen dubbels voorkomen;
- een opgave toont een 9x9 rooster met een aantal ingevulde velden:

		5	2	3	6		8	9
						2		
2								
3			4					
			8	9		3	2	7
	9		3			4	5	
		1	6	8				
					5			
		3	7					

Figuur 7.2: Sudoku spelregels: een opgave

- de oplossing is gevonden wanneer de speler het rooster correct aanvult:

1	4	5	2	3	6	7	8	9
6	3	7	9	1	8	2	4	5
2	8	9	5	7	4	1	3	6
3	1	2	4	5	7	6	9	8
4	5	6	8	9	1	3	2	7
7	9	8	3	6	2	4	5	1
5	2	1	6	8	3	9	7	4
9	7	4	1	2	5	8	6	3
8	6	3	7	4	9	5	1	2

Figuur 7.3: Sudoku spelregels: een oplossing

7.3.2 Software vereisten

- UI vereisten:
  - enkel numerieke input, beperkt tot één positie is mogelijk. Het is niet de bedoeling feedback te geven indien de gebuiker hiervan afwijkt, het is de bedoeling dat hij er niet kan van afwijken (zie ook TXTINT;
  - er is een intuitieve navigatie op basis van de pijltjes mogelijk;
  - er is kleurverschil tussen opgave en oplossing;
  - bij ingave van een foutief getal wordt dit (discreet) gekleurd.
  - de gebruiker kan de opgave niet wijzigen gedurende het oplossen
- Sudoku vereisten:
  - we kunnen opgaves ingeven en bewaren (op file), en deze later terug inlezen;
  - een opgave kan door het programma correct worden opgelost (indien er een oplossing bestaat).

7.3.3 Sudoku oplos algoritme

De recursieve oproep *zoekOplossing(Cell cell,int[,]bord)* kent twee parameters:

- *cell* is de cell die we nu wensen in te vullen. Deze stuurt de stopconditie: indien de cel die we wensen in te vullen buiten het bord valt zijn alle bord cellen ingevuld en hebben we een oplossing;
  - *Cell* is een eigen klasse (we houden rij- en kolom- waardes bij (Integers). De klasse Point werd niet gebruikt omdat deze doubles bevat);
- *bord* bevat de bordsituatie zoals ze nu gekend is. Deze situatie hebben we nodig om het correct zijn van een nieuwe zet te beoordelen. Omdat in het bord de oplossing die we zoeken verder wordt opgebouwd bij elke recursieve stap spreken we van een *accumulerende parameter*.

- het return type is *boolean*: *true* indien een oplossing wordt gevonden, *false* anders.

Het algoritme gaat als volgt:

- indien *cell* buiten het bord valt zijn alle velden ingevuld en hebben we een oplossing in *bord*;
- overloop elke mogelijke waarde voor cell en voer volgende stappen uit:
  - vul de waarde in op de nieuwe positie;
    - indien dit een Sudoku probleem veroorzaakt gaan we over naar de volgende waarde;
    - indien dit geen probleem veroorzaakt voeren we een recursieve oproep uit voor de volgende cel. Indien deze oproep *true* teruggeeft vonden we een oplossing (in bord) en returnen we zelf ook *true*. In het andere geval (de recursieve oproep levert false) was het onmogelijk om de rest van het bord in te vullen op een correcte manier;
  - verwijder de zopas gezette waarde en probeer een volgende mogelijkheid;
- indien na het overlopen van alle mogelijkheden nog geen oplossing werd gevonden moeten we false terug geven: het was niet mogelijk de Sudoku verder op te lossen: we returnen false waardoor de oproeper zijn laatste poging ongedaan maakt en een andere poging zal ondernemen.

Heb speciaal aandacht voor volgende elementen:

- een zet die niet voldoet moet terug van het bord genomen worden;
- indien een oplossing wordt gevonden mag de bord parameter (die deze oplossing bevat) niet meer gewijzigd worden.

Dit algoritme wordt ook aan bord op basis van voorbeelden ontwikkeld. Ik heb niet de ambitie backtracking volledig schriftelijke toe te lichten.

7.3.3.1 Constraint satisfaction solver

Voorgaande is niet de enige manier om Sudoku's op te lossen. Een interessante alternatieve benadering wordt beschreven in Sudoku using Microsoft solver foundation. Het onderliggende algoritme zou opnieuw backtracking kunnen zijn.

7.4 Sudoku routed events introductie

Het is de bedoeling om pijltjes navigatie op het Sudoku bord toe te laten: we vangen het *PreviewKeyDownEvent* op en voeren de eventuele gepaste navigatie uit. Elke TextBox op het Sudoku-bord (er zijn er 81) kan de bron van dergelijk event zijn: in een klassieke benadering moeten minstens 81 eventhandlers gekoppeld worden. Indien men beslist andere controls dan TextBoxen te gebruiken moet men events van deze andere controls opvangen. Om in dergelijke situaties een elegante oplossing te bieden werden *routed events* geïntroduceerd:

- een WPF-control kan zich abonneren op een *routed event* gedefinieerd op een andere klasse (zoals een DependencyObject een DependencyProperty gedefinieerd op een andere klasse kan hebben):

```
public ccSudoku9x9()
    {
    this.Background = new SolidColorBrush(Colors.Transparent);
    initSudoku9x9();
    zetContextMenu();

    this.AddHandler(TextBox.TextChangedEvent, new TextChangedEventHandler( text_changed));
    this.AddHandler(TextBox.PreviewKeyDownEvent, new KeyEventHandler(keyDown));
    }
```

Figuur 7.4: Sudoku routed events demo

- de visual tree wordt eerst van container naar child elementen doorlopen (de *preview*-events): de *tunneling* fase;
- daarna wordt de visual tree van child naar container element doorlopen (de gewone event namen): de *bubbling* fase;
- indien een eventhandler (zowel tunneling als bubbling) het event afhandelt kan de verdere eventhandling afgebroken worden (handled=true op het argument e);
  - dit afbreken is op zijn beurt terug te omzeilen;
  - door het toepassen van templates en styles kan de visual tree behoorlijk complex worden. Toch wordt hij volledig doorlopen.
- hoewel *ccSudoku9x9* noch een *TextChangedEvent* noch een *PreviewKeyDownEvent* kent kan hij toch subscriben op deze routed events. In zoverre een element in de visual tree deze events afvuurt zal de ccSudoku9x9 control aan de bubbling- en tunneling fases deelnemen;
  - gebruik *e.OriginalSource* om te achterhalen welke control het event veroorzaakte.

7.5 Async methodes: TaskCompletionSource

Indien opfrissing omtrent await/async nodig verwijs ik naar de Random topics: awaitable methodes.
Zoals in het hoofdstuk omtrent delegates werd aangekaart kan met behulp van de klasse TaskCompletionSource een awaitable methode ontwikkeld worden.

Het .NET framework 4.5 bevat veel awaitable methodes, herkenbaar aan de suffix Async (zie ook Random topics: awaitable methodes). Om dergelijke methode asynchroon uit te voeren, en de rest van de routine definitie in een callback methode voor die oproep te plaatsen volstaat het om await voor de methode oproep te plaatsen. Meer nog, de rest code volgend op de awaited

oproep wordt automatisch terug op de huidige thread geplaatst (meestal de UI-thread).

Het gebruik van een awaitable methode is gemakkelijk, het opzetten van een awaitable methode is dit minder. De volgende tekst beschrijft hoe een awaitable methode kan aangemaakt worden.

7.5.1 Task- klasses

Een awaitable methode heeft *altijd* een Task- object terug. Indien het return type van een synchrone oproep *void* is, is het return type van de asynchrone versie *Task*. Indien het return type van de synchrone uitvoering T was geweest zullen we $Task< T >$ als resultaat type gebruiken van de asynchrone oproep.

Indien de oproep wordt *awaited* mag het resultaat van de oproep (te vinden in de *Result*-paramter van de Task) als resultaat van de await gebruikt worden (de compiler lost dit op):

```
Boolean gevonden = await ZoekOplossingAsync(oplossing,0,mogelijkeKolommen);
if (gevonden)
    ToonOplossing(oplossing);
```

Figuur 7.5: Awaiting een oproep

Zowel de NQueens- als Sudoku oplosser geeft als resultaat een Boolean terug. Het resultaat type van de Async- versie zal dan ook $Task< Boolean >$ zijn.

7.5.2 TaskCompletionSource- klasses

De *await*- infrastructuur dient geïnformeerd te worden omtrent het verloop van de asynchrone uitvoering: voltooiing, onderbreking, voortgang en dergelijke meer zijn belangrijk. Omdat deze informatie niet enkel via een callback kan gecommuniceerd worden werd een nieuwe generische klasse geïntroduceerd: *TaskCompletionSource*:

- de informatie omtrent de uitvoering van een asynchrone taak wordt verzameld in een TaskCompletionSource- object;
- dergelijk object wordt door ons programma als field gedefinieerd zodat het kan gemanipuleerd worden door onze routines;

```
private void ZoekOplossingCompleted(IAsyncResult ar)
    {
    Cel[] cellen = ar.AsyncState as Cel[];
    if (cellen[cellen.Length -1] != null) //er was e
        _tcsOplossing.SetResult(true);
    else
        _tcsOplossing.SetResult(false);
    }
```

Figuur 7.6: TaskCompletionSource: SetResult

- de start van onze asynhrone methode initialiseert dit field, start het asynchrone proces (met klassieke callback) en geeft de Task- property van dit object als resultaat terug;
- het is nu de bedoeling dat onze asynchrone methode (of zijn callback) na uitvoering dit object als completed of cancelled gaat instellen. De compiler heeft code gegenereerd die hierop inpikt om dan de logica die wachtte verder uit te voeren.

Wellicht kan een codevoorbeeld dit verduidelijken:

```
private delegate Boolean ZoekOplossingDelegate(Cel[] vorigeZetten, int
private TaskCompletionSource<Boolean> _tcsOplossing;
private Task<Boolean> ZoekOplossingAsync(Cel[] vorigeZetten, int huidi
    {
    _tcsOplossing = new TaskCompletionSource<Boolean>();

    ZoekOplossingDelegate del = new ZoekOplossingDelegate(ZoekOplossin
    del.BeginInvoke(vorigeZetten,huidigeRij,mogelijkeKolommen,ZoekOplo

    return _tcsOplossing.Task;
    }
```

Figuur 7.7: TaskCompletionSource demo

- ZoekOplossingDelegate is de delegate klasse om het zoeken asynchroon te programmeren;
- tcsOplossing is het resultaat van de awaitable methode. Ze wordt als field gedefinieerd zodat mijn logica deze kan aansturen;
- ZoekOplossingAsync is de awaitable methode (herkenbaar aan het return-type en de suffix van de methode). Deze methode zal intern een asynchrone oproep starten en geeft onmiddelijk een Task-object terug. De compiler genereert code die in de oproeper de rest van de logica uitvoert nadat deze task voltooid wordt. Het voltooien van de Task gebeurt door het resultaat in te stellen op het *TaskCompletionSource*- object waar de Task onderdeel van uitmaakt;

7.6 Het MiniMax algoritme (geen examen stof)

Backtracking wordt onder andere gebruikt als Artificiële Intelligentie algoritme bij het ontwikkelen van (turn-based) games. Het *MiniMax* algoritme is hiervan een voorbeeld. De Engelstalige wikipedia-entry voor MiniMax vind ik inhoudelijk sterker dan de Nederlandstalige.

7.6.1 MiniMax benadering voor vier op een rij

Het MiniMax algoritme realiseert volgende doelstelling: indien de speler aan zet niet wint met zijn volgende zet, dan minimaliseert hij de kansen van de andere speler. De andere speler zal bij het overlopen van zijn eigen mogelijkheden, indien hij niet direct wint, veronderstellen dat zijn tegenstander (speler 1) geen fouten maakt, en diens score maximaliseren, vandaar de naam van het algoritme. Omdat we bordsituaties met elkaar vergelijken (om een beste of slechtste situatie te kunnen kiezen) zullen we een BordWaarde- functie nodig hebben die de waarde van een bordsituatie weergeeft.

7.6.1.1 BordWaarde

- de parameters zijn die waardes nodig om een bordsituatie te beschrijven;
- het resultaat is een numerieke waarde, zodat vergelijkingen mogelijk worden:
  - +1: indien de speler aan zet uitspeelt;
  - -1: indien de tegenspeler uitspeelt;
  - 0: anders.
- de BordWaarde- functie zal intens gebruikt worden: elke mogelijke stap zal geëvalueerd worden door deze functie. Het is dan ook belangrijk om deze functie zo efficient mogelijk te maken.

Het is contra- intuitief om de bord- waarde te beperken tot 3 waardes: winst, verlies of onbeslist. Immers, wanneer we zelf vier op een rij spelen streven we bijvoorbeeld naar een situatie waar twee aansluitende rijen in een kolom tot winst leiden (zie kolom 3 in onderstaande figuur). In onze beoordeling van een bord is dit duidelijk een zeer goede situatie. MiniMax gewijs zal de AI, indien hij drie zetten vooruit *denkt* (rekent) ook merken dat in elke mogelijke invulling van een dergelijke situatie we een winnende zet kunnen realiseren, wat van dergelijke bord- situatie een winnende situatie maakt (+1). In onderstaande figuur werd de situatie in kolom 3 hierdoor als winnend herkent (bij de vorige zet) omdat de AI minstens drie zetten vooruit denkt:

Figuur 7.8: 4 op een rij: bordwaarde

7.6.1.2 BesteZet

De functie BesteZet berekent voor een gegeven bord situatie de best volgende zet voor een speler. In een vier op een rij context is het resultaat van de oproep een kolom nummer (int).

- het resultaat van de oproep is het kolomnr waarin de speler zijn fiche moet plaatsen. De parameters van de oproep worden bepaald door de nood van de hieronder geformuleerde logica;
- één van de argumenten van de oproep, *depth* geeft aan hoeveel stappen de computer vooruit mag denken. Hoe meer stappen de computer vooruit denkt, hoe langer de berekening zal duren.
- de speler aan zet (de maximizing speler) overloopt elke mogelijke kolom en voert volgende zaken uit:
  - plaats een fiche in deze kolom;
  - indien vier op een rij werd gevonden hebben we een winnende kolom: *return kolomnr*
  - indien geen directe winst wordt de waarde van deze kolom bepaalt als MiniMax(maximizingSpeler, minimizingSpeler, depth -1, ..).
  - indien de MiniMax- waarde kleiner is dan het huidige minimum wordt deze zet het nieuwe minimum (bewaar zowel de kolom als het minimax resultaat);
  - verwijder deze fiche terug
- de kolom met de kleinste MiniMax waarde (de slechtst mogelijke zet voor de tegenstander) wordt als resultaat teruggegeven.

Om wat random zetten aan het algoritme toe te voegen kunnen we volgende zaken aanpassen:

- we houden niet enkel de eerste beste zet bij, maar alle zetten die de huidige beste waarde als resultaat hebben;
- bij het teruggeven van de beste kolomwaarde wordt een random kolomnummer uit de lijst gekozen.

7.6.1.3 MiniMax methode

De Engelstalige wikipedia-entry voor MiniMax beschrijft het MiniMax algoritme als volgt:

```
function minimax(node, depth, maximizingPlayer)
    if depth = 0 or node is a terminal node
        return the heuristic value of node
    if maximizingPlayer
        bestValue := −∞
        for each child of node
            val := minimax(child, depth - 1, FALSE))
            bestValue := max(bestValue, val);
        return bestValue
    else
        bestValue := +∞
        for each child of node
            val := minimax(child, depth - 1, TRUE))
            bestValue := min(bestValue, val);
        return bestValue
```

Figuur 7.9: Minimax algoritme

- een *terminal node* is een zet die het spel afrondt: één van beide spelers realiseert vier op een rij. We veronderstellen dat enkel de speler die nu een fiche plaatst kan uitspelen, en dat zijn huidige zet deel uitmaakt van de oplossing (dat wil zeggen: we veronderstellen dat er zijn geen andere vier op een rij oplossingen zijn).
- u moet weten wie de maximizing speler is opdat u weet of de speler aan zet de mogelijkheden van zijn tegenstrander minimaliseert of maximaliseert;
- die MiniMax algoritme overloopt *alle* mogelijke zetten (tot eindspelsituaties) voor beide spelers, wat in veel spelsituaties een gigantisch aantal mogelijkheden betekent. Het beperken van die set mogelijkheden, zonder de beste oplossing te verliezen is dan ook zeer wenselijk.

7.6.2 Minimax optimalisaties

7.6.2.1 Alfa-Beta pruning

Indien u het MiniMax algoritme correct kon implementeren (geen onderdeel van de cursus), dan zal u wellicht geïnteresseerd zijn in Alfa-Beta pruning voor de MiniMax zoek boom. De uitleg en het bijhorende voorbeeld onderaan de webpagina kan ik niet verbeteren.

Een nadeel van maximaal pruning is het verliezen van de alternatieve beste zetten. Dit kan als gevolg hebben dat de AI zijn zetten nogal éénzijdig in het speelveld plaatst: indien de kolommen van links naar rechts worden doorgelopen zullen de eerste beste oplossingen zich altijd links ten opzichte van hun alternatieven bevinden. Omdat enkel de eerste beste oplossing wordt weerhouden wordt het speelveld vooral links ingevuld.

7.6.2.2 Multi- threading

Op een multi core toestel zal het single- threaded berekenen van de beste zet geen optimaal gebruik maken van de beschikbare rekenkracht. Het op een aparte thread plaatsen van de BesteZet berekening maakt deze berekening zelf niet multi threaded. Het multithreaded maken van vier op een rij lijkt voor de hand liggend, maar moet ik zelf nog eens uitwerken. Bij het multithreaded maken moet rekening worden gehouden met volgende elementen:

- de verschillende threads mogen elkaars situatie niet beïnvloeden. Elke variabele die gewijzigd wordt moet dan ook een copy van de oorspronkelijke waarde zijn.
  - veronderstel dat elke kolom-invulling in een aparte thread wordt gestart. Voor de eerste zet van de eerste speler worden dan onmiddellijk de acht eerste kolommen ingevuld. Indien ze op een gemeenschappelijk veld worden geplaatst heb ik onmiddellijk *acht* op een rij gerealiseerd. Bovenstaande redenering is van toepassing op *elke* variabele die door een thread zal worden aangepast;
  - bij het multithreaded maken van de oplossing worden verschillende deelzoekbomen parallel uitgevoerd. Dit conflicteert met de winst bekomen door de alfa- beta pruning waarbij een nieuwe deelboom pas wordt verwerkt nadat de beste oplossing uit de vorige deelboom werd berekend.
    - wellicht is het alfa-beta prunen van een multithreaded oplossing wel realiseerbaar, maar het is maar de vraag hoe groot de winst nog zal zijn. Indien de multithreaded oplossing pas even snel is als de pruned singled threaded oplossing hebben we gewoon resources onnodig verbruikt. Hier is nog ruimte voor wat onderzoek :).

7.7 Backtracking opgaves

- ontwikkel een N-queens oplosser;
  - Extended WPF Toolkit bevat tal van interessante WPF-controls, onder andere verschillende up/down controls. Wellicht kan u een aantal hiervan gebruiken bij het opzetten van uw oplossing;
  - voorzie een synchrone uitvoering, en demonstreer het blokkeren van de interface;
  - voorzie een asynchrone uitvoering (met cancel- mogelijkheid) en demonstreer de responsiveness van de interface (bijvoorbeeld door de cancel uitvoering);
  - voorzie een tweede asynchrone uitvoering die gebruikt maakt van het async/await mechanisme (zie cursustekst).
- ontwikkel een Sudoku oplosser.
  - voorzie een synchrone uitvoering. Het resultaat wordt zo snel bekomen dat een asynchrone uitvoering weinig zin heeft.

Jef Daels

Hoofdstuk 8

Dependency properties

8.1 Dependency properties: duiding

In een klassieke object georiënteerde omgeving definieert en implementeert een klasse de eigenschappen van haar instanties. In .NET wordt hiertoe een property- syntax gebruikt (we spreken hier van een CLR-property), in JAVA gebruikt men getter- en setter- routines. Zo zal de Text- property van de klasse TextBox in de TextBox-klasse zelf worden geïmplementeerd.

WPF introduceert de extra mogelijkheid om een property, gedefinieerd in klasse A, te koppelen aan instanties van klasse B. Dit vereenvoudigt sterk het opzetten van nieuwe infrastructuren: indien een Grid-klasse nood heeft aan het toekennen van *row* en *column* waardes aan subcontrols, dan kan de *Grid* klasse deze properties zelf introduceren. Deze dependency properties spelen ook een cruciale rol in WPF's databinding en animatie.

In praktijk kan u het dependency property systeem zien als een dictionary waarin voor objecten extra waardes worden bijgehouden. De infrastructuur laat toe om bij het instellen van de waardes custom code uit te voeren.

Attached dependency properties zijn XAML-georiënteerde dependency properties: om in XAML bruikbaar te zijn moet een dependency property een CLR wrapper hebben (die at run-time trouwens gebypassed wordt ..).

8.1.1 Relevante namespaces

- *System.Windows*
- *System.ComponentModel*
- *System.Windows.Media*
- *System.Windows.Data*

8.2 F1URL: een help infrastructuur

We zullen een F1-gebaseerd help systeem opzetten voor de UI-elementen in onze toepassing(en). Hiertoe wensen we de mogelijkheid te hebben om voor een willekeurige control een URL met de gewenste help bij te houden. Wanneer de gebruiker F1 klikt zal de webpagina van de control waarboven de cursor zich bevindt getoond worden. Indien de controls genest zijn zullen de eventuele verschillende pagina's getoond worden. Momenteel wordt in het .NET framework niet de mogelijkheid voorzien om declaratief een URL te koppelen aan een control (u kan natuurlijk zelf een dictionary systeem opzetten).

8.2.1 F1URL gebruik in XAML

```
<Window x:Class="URLDemo.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525"
        xmlns:src="clr-namespace:URLDemo"
        x:Name="window"
        src:F1URL.URL="abc"
        >
    <src:F1URL x:Name="grid">
        <Button Height="175" Name="btnOuter"
                HorizontalAlignment="Left" Margin="128,69,0,0"  VerticalAlignment="Top" Width="262"
                src:F1URL.URL="www.google.com" >
            <Button Content="Button" Height="67" HorizontalAlignment="Left"
                    x:Name="btnInner" VerticalAlignment="Top" Width="121"
                    src:F1URL.URL="http://www.howest.be" />
        </Button>
    </src:F1URL>
</Window>
```

Figuur 8.1: Attached dependency property in XAML

In bovenstaande XAML-code zijn volgende elementen van belang:

- de namespace *src* hebben we zelf toegevoegd (gebruik de intellisense!);
- *src:F1URL* is een klasse die erft van *Grid*. De controls waarop de F1URL benadering moet werken zullen binnen dergelijke grid geplaatst worden (de *F1URL*- control zal het *KeyUp* event opvangen voor alle geneste subcontrols);
- de (attached) dependency property *F1URL.URL* wordt ingesteld voor het window en twee (geneste) buttons. De geneste buttons introduceren onmiddellijk een extra complexiteit: indien we F1 drukken als de inner button geselecteerd is, wensen we dan de help van de inner of van de outer button?

Wanneer we de Button- klasse erop nazien merken we dat er geen URL-property gekend is. Hoe werd het dan mogelijk om deze property toch in te stellen?

Jef Daels

8.2.2 F1URL definitie

```
public class F1URL : System.Windows.Controls.Grid
    {
    public static readonly DependencyProperty URLProperty =
        DependencyProperty.RegisterAttached("URL", typeof(string), typeof(F1URL), new UIPropertyMetadata(null));

    //[System.Windows.AttachedPropertyBrowsableForType(typeof(UIElement))]
    [System.ComponentModel.Browsable(true)]
    public static string GetURL(UIElement target)
        {
        return (string)target.GetValue(F1URL.URLProperty);
        }

    public static void SetURL(UIElement target, string value)
        {
        target.SetValue(F1URL.URLProperty, value);
        }
```

Figuur 8.2: Attached dependency property definitie ($C^{\#}$)

Bemerk in bovenstaande code volgende zaken:

- een dependency property is een *static* field van het type *DependencyProperty*. De naam van het field eindigt met het woord *Property*;
- een nieuwe dependency property instantie wordt aangemaakt door middel van een *register(Attached)*- oproep met volgende argumenten:
  - de naam in de registratie is identiek aan de fieldnaam, met weglating van de suffix *Property*;
  - het property type definieert het type van de property (een tekst- property heeft hiervoor als parameter *typeof(string)*);
  - het ownertype definieert de klasse die als prefix voor de property zal gebruikt worden: *F1URL*.URL (typeof(F1URL) is hierin het owner type);
  - extra informatie omtrent het verwerken van de property waardes.
- we voorzien de dependency property van (static) Get- en Set- wrapper routines (de hoofdletters *G* en *S* zijn belangrijk!). De Get- en Set- namen zijn (op de drie-letter prefix na) identiek aan de naam gebruikt in de register methode. Zonder deze wrappers is de dependency property niet gekend in XAML.
  - merkwaardig genoeg zal XAML de wrappers volledig bypassen, zoals beschreven in XAML Loading and Dependency Properties;
- indien we de Get- wrapper voorzien van een *Browsable*- attribuut zal deze dependency property ook getoond worden in de property list in Blend (in Visual Studio 2010 wordt de property getoond indien hij in de XAML-code aanwezig is);
- de snippet *propdp* is standaard in Visual Studio aanwezig om de dependency property definiërende code te genereren(*propa* voor attached dependency properties).

8.2.3 F1URL infrastructuur

We definieerden de dependency property *F1URL.URL* om een help infrastructuur op te zetten. Indien een gebruiker F1 duwt wensen we een help-pagina te tonen. Om het key-up event van een willekeurige control op te vangen gaan we als volgt te werk:

- we maken een eigen Grid-klasse versie waarbinnen alle controls die F1URL-ondersteuning nodig hebben geplaatst zullen worden;
  - de hier voorgestelde oplossing werkt enkel voor controls binnen onze eigen Grid-klasse. Deze beperking is overbodig wanneer we echte behaviors zullen ontwikkelen;
  - de huidige opzet is geschikt om bubbling (routed) events toe te lichten;
- de container control zal het bubbling *keyup*-event opvangen en actie ondernemen om de nodige help-topics te tonen:

```
public class F1URL : System.Windows.Controls.Grid
    {
    "dep prop URL"
    public F1URL() : base()
        {
        this.KeyUp += new System.Windows.Input.KeyEventHandler(URLF1_KeyUp);
        }

    void URLF1_KeyUp(object sender, System.Windows.Input.KeyEventArgs e)
        {
        if (e.Key == System.Windows.Input.Key.F1)
            verwerkF1(e.OriginalSource as FrameworkElement);
        }

    private void verwerkF1(FrameworkElement fwe)
        {
        if (fwe == null) return;
        string naam = fwe.Name;
        string url = (string) fwe.GetValue(F1URL.URLProperty);
        if (url != null)
            toonURL(url,naam);
        verwerkF1(fwe.Parent as FrameworkElement);
        }

    private void toonURL(string URL,string naam)
        {
        try
            {
            System.Diagnostics.Process.Start(URL);
            }
        catch (Exception ex)
            {
            }
        }
    }
```

Figuur 8.3: F1 infrastructuur ($C^{\#}$)

- de naam van het frameworkelement (een control parent klasse) is aanwezig omwille van debug-redenen;
- de *Process*-klasse start automatisch uw favoriete browser (bemerk dat *Process* ook het geschikte programma zal starten indien het argument geen URL zou zijn);

- de infrastructuur verwerkt alle geneste controls.

Uiteraard is deze eenvoudige benadering niet helemaal naar wens: zo willen we help kunnen opvragen omtrent een knop zonder(!) dat we deze dienen aan te klikken: het aanklikken start namelijk het proces waarvoor u de help wenst op te vragen, dit is zeker niet wenselijk. De infrastructuur moet rekening houden met de positie van de cursor en niet met de geselecteerde control. De lesdemo illustreert dit en maakt hiertoe gebruik van hit- testing om de control onder de cursor op te zoeken.

8.3 Dependency properties: technisch

- een dependency property is een static DependencyProperty- field in een klasse;
- een dependency property heeft een *owner type*. Dit is niet noodzakelijk de klasse waarbinnen de property wordt gedefinieerd. Een dependency property kan trouwens meerdere owner types hebben (we maken dan gebruik van de addOwner methode van de DependencyProperty klasse).
- een attached dependency property is een dependency property bruikbaar via XAML. Opdat XAML de property zou kennen moeten de naamgevingsregels gevolgd worden!
- bij het registreren van een dependencyproperty kan optioneel extra informatie worden meegegeven:
  - een default waarde;
  - verschillende routines (allen in de vorm van delegates):
    - propertyChangedCallback: wordt uitgevoerd wanneer de property waarde wijzigt (niet noodzakelijk bij elke toekenning!);
    - coerceValueCallback: wordt gebruikt om een eventuele nieuwe waarde om te vormen tot een andere waarde;
    - validateValueCallback: valideert de geldigheid van de nieuwe waarde voor deze klasse (niet in combinatie met de andere state elementen van het target object waaraan de value wordt toegekend). Het boolse resultaat wordt onmiddellijk gebruikt in de XAML-editor om de eventuele incorrecte waarde aan te geven. Deze routine wordt uitgevoerd *na* de coerce- routine;
- het definiëren van een dependency property is niet voldoende om deze property voor een object te kunnen instellen: dependencyproperties zijn enkel instelbaar op instanties van klasses die erven van DependencyObject!

Het is een aanrader om als delegate oefening eens een uitgewerkte URL- attached dependency property aan te maken met volgende elementen:

- een URL is slechts geldig indien hij start met *http://* (hoofdletterongevoelig);
- voor elke TextBox control zal de url *http://www.howest.be* worden omgevormd tot *http://www.nmct.be*.

Stel in XAML-code een aantal URL's in om uw implementatie te testen. Voorzie in uw testen volgende situaties:

- voorzie een URL-waarde die niet start met de gevraagde prefix, en observeer het resultaat in XAML en bij het opstarten;
- voorzie een textbox van een om te vormen URL;
- pas de coerce routine aan zodat de nieuwe waarde niet voldoet aan de validate routine en observeer het resultaat.

8.4 Dependency property XAML binding

Via XAML-code kunnen we enkel dependency properties *binden*. Indien we proberen een binding op te zetten voor een gewone (CLR) property, dan krijgen we bij uitvoering volgende foutmelding:

```
<src:ccNQueens Grid.Row="1" x:Name="grdOplossing"
               DimensieCLR="{Binding Value, ElementName=iudQueens}"/>

id>
>
```

XamlParseException occurred
A 'Binding' cannot be set on the 'DimensieCLR' property of type 'ccNQueens'. A 'Binding' can only be set on a DependencyProperty of a DependencyObject.

Figuur 8.4: CLR binding error

8.5 Dependency property syntax

Het instellen van een dependency property kan in $C^{\#}$op twee manieren gecodeerd worden:

- *target.setValue(ownertype.depprop, value)*: het object gebruikt de geërfde methode setValue om een dependencyproperty waarde te bewaren;
- *ownertype.Setdepprop(target,value)*: de static setter routine wordt gebruikt;

```
anWidth.SetValue(Storyboard.TargetNameProperty,btnInner.Name);
anWidth.SetValue(Storyboard.TargetPropertyProperty, new PropertyPath( Button.WidthProperty));
```

Figuur 8.5: Dependency property syntax 1 ($C^{\#}$)

```
Storyboard.SetTargetName(anWidth, btnInner.Name);
Storyboard.SetTargetProperty(anWidth, new PropertyPath(Button.WidthProperty));
```

Figuur 8.6: Dependency property syntax 2 ($C^{\#}$)

Er bestaan uiteraard gelijkaardige notaties voor het opvragen van de dependency property. Beide notaties komen aan bod gedurende het bespreken van de animaties.

8.6 Dependency properties en animaties

WPF ondersteunt een animatie infrastructuur (niet de topic hier) gebaseerd op dependency properties: het wijzigen van een dependency property met impact op de visualisatie zal deze visualisatie onmiddellijk wijzigen. Daarbovenop komen een aantal (animatie-) klasses die gebruikt kunnen worden om waardes van start- waarde naar een eind- waarde te laten evolueren gedurende een tijdsinterval (met easing). In combinatie met Blend maakt dit het vlot opzetten van tijdlijn animaties en storyboards mogelijk.

In dit cursusdeel zullen we de animaties implementeren door middel van $C^{\#}$- code. Door Blend gegenereerde code is XAML code. Het is *niet* de bedoeling om een impliciete voorkeur te laten blijken.

8.6.1 DoubleAnimation

De breedte van een control is een double waarde. De DoubleAnimation klasse voorziet de mogelijkheid om een double waarde geleidelijk te evolueren tusen een start- en eindwaarde. Indien men dan de breedte- eigenschap van een control koppelt aan deze double- evolutie bekomt men een animatie:

```
private void animeerBreedte()
    {
    DoubleAnimation anWidth = new DoubleAnimation(btnInner.Width, btnInner.Width * 1.1,
        new Duration(new TimeSpan(0, 0, 2)));
    anWidth.SetValue(Storyboard.TargetNameProperty,btnInner.Name);
    anWidth.SetValue(Storyboard.TargetPropertyProperty, new PropertyPath( Button.WidthProperty));
    anWidth.AutoReverse = true;
    anWidth.RepeatBehavior = RepeatBehavior.Forever;

    Storyboard sb = new Storyboard();
    sb.Children.Add(anWidth);
    sb.Begin(this);
    }
```

Figuur 8.7: Breedte animatie ($C^{\#}$)

Op basis van bovenstaande code formuleren we volgende opmerkingen:

- de DoubleAnimation instantie zelf weet niet op welke eigenschap van welk object ze zal worden toegepast;
- het koppelen van een DoubleAnimation aan een eigenschap van een control gebeurt door middel van twee dependency properties, gedefinieerd in de StoryBoard klasse:
  - *StoryBoard.TargetNameProperty* wordt gebruikt om de naam te bepalen van het element (de control) waarvan een eigenschap geanimieerd zal worden;
  - *StoryBoard.TargetPropertyProperty* wordt gebruikt om de dependencyproperty die geanimeerd zal worden te identificeren op basis van zijn PropertyPath.
- animaties worden (hier) als onderdeel van een StoryBoard uitgevoerd;
- het starten van een StoryBoard heeft nood aan een visueel element waarbinnen de te animeren controls worden gezocht.
  - in *WinRT*-toepassingen is het niet nodig de *Begin*- methode van een *StoryBoard* van parameters te voorzien.

8.6.2 ColorAnimation

Het animeren van de achtergrondkleur van een control gebeurt op gelijkaardige wijze maar is verschillend op volgende punten:

- de achtergrond van een control is een Brush (en geen Color!). Het animeren van de achtergrondkleur gebeurt dan ook door de kleur van een (SolidColor)Brush te animeren (we hebben een ColorAnimation- class, en geen SolidColorBrushAnimation class);
- het PropertyPath, nodig om een achtergrondkleur te animeren bestaat uit twee stukken: het eerste stuk kiest de Background property, het tweede stuk kiest daarvan de SolidColorBrush.Color property. De code zal enkel werken indien de background een SolidColorBrush waarde bevat. Indien dit niet het geval is (de achtergrond is niet ingesteld, of is een ander soort Brush), dan zal de achtergrond uiteraard niet geanimeerd worden (de gezochte eigenschap is niet aanwezig). Dit genereert trouwens *geen* uitvoeringsfout.

```
private void animeerBackground()
    {
    ColorAnimation anKleur = new ColorAnimation(Colors.Red,new Duration(new TimeSpan(0,0,2)));
    Storyboard.SetTargetName(anKleur, btnInner.Name);
    Storyboard.SetTargetProperty(anKleur,new PropertyPath("(Panel.Background).(SolidColorBrush.Color)"));
    anKleur.AutoReverse = true;
    anKleur.RepeatBehavior = RepeatBehavior.Forever;
    Storyboard sb = new Storyboard();
    sb.Children.Add(anKleur);
    sb.Begin(this);
    }
```

Figuur 8.8: Background animatie

8.6.3 Transformatie animaties

Het tranformeren van een figuur bestaat veelal uit drie onderdelen: een schaling, een rotatie en een skewing. Voor elk van deze onderdelen bestaat er een transformatie klasse. Verschillende transformaties kunnen worden gebundeld tot een transformatiegroep:

```
private void transformeer()
    {
    ScaleTransform scale = new ScaleTransform(0.75, 1.1);
    RotateTransform rot = new RotateTransform(90);
    SkewTransform skew = new SkewTransform(30, 30);
    TransformGroup tg = new TransformGroup();
    tg.Children.Add(scale);
    tg.Children.Add(rot);
    tg.Children.Add(skew);

    btnOuter.RenderTransform = tg;
    }
```

Figuur 8.9: Transformatie van een Button ($C^{\#}$)

- in veel gevallen is het gemakkelijker om een CompositeTransform te gebruiken in plaats van een transformatiegroep.
- bovenstaande code is nog geen animatie: er wordt een eenmalige transformatie toegepast op een control die daarna onveranderd blijft;

Een animatie zou er in bestaan om bijvoorbeeld de schaling en/of de draaihoek te animeren. Dit zijn properties van de zopas gedefinieerde objecten rot en scale. Deze objecten zijn niet bij naam gekend door de animatie infrastructuur (in de vorige animaties kon het te animeren object (een control) via zijn naam geïdentificeerd worden). Om properties van deze nieuwe objecten toch te kunnen animeren zullen we een naam voor deze objecten registreren. Na registratie kunnen ze via hun naam in een TargetNameProperty context gebruikt worden:

- in een *WinRT* toepassing kunnen we een object animeren zonder zijn naam te kennen door middel van de *TargetProperty*- dependency property van de StoryBoard klasse.


```
private void transformeer()
    {
    ScaleTransform scale = new ScaleTransform(0.75, 1.1);
    RotateTransform rot = new RotateTransform(90);
    SkewTransform skew = new SkewTransform(30, 30);
    TransformGroup tg = new TransformGroup();
    tg.Children.Add(scale);
    tg.Children.Add(rot);
    tg.Children.Add(skew);

    btnOuter.RenderTransform = tg;

    this.RegisterName("myrot", rot);
    this.RegisterName("myscale", scale);

    DoubleAnimation anAngle = new DoubleAnimation(rot.Angle, rot.Angle + 360,new Duration(new TimeSpan(0,0,4)));
    anAngle.SetValue(Storyboard.TargetNameProperty, "myrot");
    anAngle.SetValue(Storyboard.TargetPropertyProperty, new PropertyPath(RotateTransform.AngleProperty));
    anAngle.RepeatBehavior = RepeatBehavior.Forever;

    Storyboard sb = new Storyboard();
    sb.Children.Add(anAngle);
    sb.Begin(this);
    }
```

Figuur 8.10: Transformatie animatie ($C^{\#}$)

8.6.4 Animaties en XAML

In veel gevallen is het gemakkelijker om animaties aan te maken met behulp van een design tool (*Blend*) dan via programmeer code. De door Blend gegenereerde code is XAML-code die buiten het bestek van dit hoofdstuk valt.

8.6.5 $C^{\#}$of XAML animaties?

Het animeren via code wordt naar mijn aanvoelen minder aangemoedigd dan het animeren via XAML (met behulp van tools zoals Blend). Toch meen ik dat een aantal situaties te herkennen zin waarbij $C^{\#}$- animatie een elegante oplossing kan bieden:

- animatie van dynamisch toegevoegde controls;
- animaties waarvoor de toe te passen waardes afhankelijk zijn van de run-time situatie en niet at-design time gekend zijn. Voorbeelden hiervan zijn:
  - het zweven van een fiche naar een speelveld op een spelbord;
  - het proportioneel verschuiven van elementen binnen een control, die niet altijd even groot moet zijn;

Jef Daels

- een aantal basis- animaties (zoals bv. opacity) heb ik als extention method gedefinieerd op de control klasse. Het lijkt me dan ook weinig zinvol om deze animatie in XAML te definiëren.

Bovenstaande situaties zijn implementeerbaar via XAMl, maar behoeven dan meestal andere extra $C^{\#}$-code elementen zoals TypeConvertor of extra properties.

8.7 DependencyProperties in XAML: TypeConverters

Indien we in XAML een dependency property instellen moeten we ons beperken tot tekst. Dit is bruikbaar voor tekstwaardes, maar niet indien het resultaat type een niet-tekst type is. Om dit toch mogelijk te maken worden typeconvertoren gebruikt. Deze zijn in staat om op basis van een string een instantie van het property type te construeren (een vorm van deserialisatie). WPF bevat reeds een aantal typeconvertoren voor frequent voorkomende conversies (getallen, kleuren, ..) maar in sommige gevallen is het toch nodig om ook zelf een typeconverter te implementeren.

8.7.1 IValueConverter interface

Deze interface bevat een *Convert* en *ConvertBack* methode (in onderstaand voorbeeld is enkel de Convert van belang).

Het *ValueConversion* attribuut wordt gebruikt om het systeem te informeren omtrent de geïmplementeerde typeconversies. Onderstaande code codeert een converter die tekst omvormt tot een *LineairGradientBrush*. Er wordt verondersteld dat de twee kleurnamen in het Engels, en hoofdlettergevoelig geschreven worden:

```
[System.Windows.Data.ValueConversion(typeof(string),typeof(System.Windows.Media.LinearGradientBrush))]
public class LinearGradientConverter : System.Windows.Data.IValueConverter
    {
    public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
        {
        string tekst = value as string;
        if (tekst == null) return null;

        string[] kleuren = tekst.Split(new char[]{','});
        BrushConverter bc = new BrushConverter();
        Color kleur1 = ((SolidColorBrush) bc.ConvertFrom(kleuren[0])).Color;
        Color kleur2 = ((SolidColorBrush)bc.ConvertFrom(kleuren[1])).Color;
        return new LinearGradientBrush(kleur1, kleur2, 45);
        }

    public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
        {
        throw new NotImplementedException();
        }
    }
```

Figuur 8.11: ValueConverter definitie ($C^{\#}$)

In XAML kunnen we dan volgende code noteren:

```
<Window.Resources>
    <src:LinearGradientConverter x:Key="lgc"></src:LinearGradientConverter>
    <s:String x:Key="redgreen">Red,Green</s:String>
</Window.Resources>
<src:F1URL x:Name="grid">
    <Button Height="129" Name="btnOuter"
            HorizontalAlignment="Left" Margin="128,69,0,0"  VerticalAlignment="Top" Width="262"
            src:F1URL.URL="http://www.google.com"
            Background="{Binding Source={StaticResource ResourceKey=redgreen}, Converter={StaticResource ResourceKey=lgc}}" >
    </Button>
```

Figuur 8.12: ValueConverter XAML ($C^{\#}$)

Hier wordt de achtergrond van een Button (een Brush dus) ingesteld door middel van een tekst (Red,Green) en een typeconverter. Het resultaat is een LineairGradientBrush.

Jammer in de XAML notatie is dat de tekst-waarde momenteel als static resource gedefinieerd wordt (misschien kan dit anders?)

8.7.2 Default dataconverters

Veelal lijkt de convertie van tekst in XAML naar een overeenkomstig object spontaan te gebeuren, zonder expliciete vermelding van de nodige type converter. Om dit te realiseren wordt een klasse voorzien van een default typeconverter (door middel van het TypeConverterAttribute attribuut). De typeconverter klasse wordt jammer genoeg anders opgebouwd dan de *ValueConverters* die eerder aan bod kwamen. Om een klasse als impliciete typeconverter te kunnen gebruiken gaan we als volgt te werk:

- de klasse die de conversie zal uitvoeren *moet* erven van de klasse *TypeConverter* (de interface *IValueConverter* is hier onbelangrijk);
- het is belangrijk om minstens twee methodes te overriden:
  - *CanConvertFrom* die default *false* terug geeft die aangeeft of het gevraagde brontype wel geschikt is om te converteren (in XAML is het brontype *String*);
  - *ConvertFrom* is de methode die de effectieve conversie uitvoert.

Wanneer een klasse een default typeconverter kent die tekst kan omvormen tot een instantie van de klasse, dan is het niet langer nodig om in de XAML-code een expliciete converter te voorzien.

8.7.2.1 TryParse

Wanneer ik een conversie van tekst naar een type X moet coderen zal ik het type X voorzien van een *TryParse* methode:

- het resultaat type is Boolean: true indien de conversie lukt, false anders;

- de eerste parameter bevat de tekstwaarde die geconverteerd moet worden;
- de tweede parameter (een referentie parameter) bevat het resultaat na de conversie (ongewijzigd indien de conversie niet lukt).

Het gebruik van *TryParse* is in .NET reeds op vele plaatsen terug te vinden (Integer, Double, DateTime, IPAdress, ..). Het implementeren van dergelijke methode is vooral een conventie, en zeker geen technische vereiste.

8.8 Dependency properties en OnRender

Indien een dependencyproperty belangrijk is voor het visuele aspect van de dependency objecten waarop de property wordt toegepast dan kan dit in de meta informatie worden opgenomen (bij het registreren van de dependency property). Indien dit het geval is zal het wijzigen van de dependencywaarde de *OnRender*- methode van het object waarvoor de dependecyproperty wijzigt triggeren. Dit op zijn beurt heeft als gevolg dat u het hertekenen niet moet oproepen via de propertychanged callback methode. Indien u dit wenst uit te pluizen (helemaal niet moeilijk) kan u zoeken op *FrameworkPropertyMetadata.AffectsRender* en *FrameworkPropertyMetadata.AffectsMeasure*.

8.9 Veel voorkomende fouten

- niet volgen van de naamgevings conventies (ook hoofdletters!);
- *null* als default waarde van een value type;

8.10 Extra lectuur

- Wie een interessant voorbeeld met vele depencendy properties wenst na te lezen verwijs ik graag door naar Render Text On A Path With WPF
- het aanmaken van een eigen dependency property wordt beschreven in Custom Dependency Properties
- een goede technische beschrijving omtrent attached dependency properties

8.11 Dependency property: opgaves

- animeer via code een control:
  - zijn afmetingen (breedte en grootte);
  - zijn achtergrondkleur;
  - roteer de control;
- maak een extention method waarmee u de opacity van een control (elke control) kan animeren: FadeInOut(double opacity);
- implementeer de *F1URL* infrastructuur en demonstreer dat deze werkt.
- voorzie de NQueens toepassing van een *Dimensie*- dependency properties, en bind deze aan de IntegerUpDown control zodat de code-behind leeg wordt;

```
<src:ccNQueens Grid.Row="1" x:Name="grdOplossing"
               src:ccNQueens.Dimensie="{Binding Value, ElementName=iudQueens}"  />
```

Figuur 8.13: XAML binding dependency property

Jef Desnaels

Hoofdstuk 9

Templated controls

Eventuele oefening: een ccDatePicker

9.1 Templated controls: duiding

Een betrachting in moderne WPF toepassingen is het scheiden van logica en visualisatie (de wenselijkheid hiervan hangt af van de situatie). Dit wordt in veel gevallen gerealiseerd door het gebruik van XAML (usercontrols, templates, styles, ..) die gebonden wordt aan (dependency) properties gevonden in de logica (zie ook . Dit hoofdstuk behandelt extra mogelijkheden (bovenop het binden van properties) om interactie tussen beide lagen mogelijk te maken.

- de term *templated controls* duidt hier op die controls die in de visualisatie van het dataelement zijn opgenomen.
- dit hoofdstuk is geen pleidooi voor of tegen de scheiding tussen logica en view. Het al dan niet wenselijk zijn van die scheiding is projectafhankelijk, en de factoren zijn niet enkel puur technisch: vraag van de klant, huisstijl, beschikbare tijd, .. zijn evenzeer elementen die van belang zijn.

In dit hoofdstuk krijgt u zeer weinig code te zien. Het is dan ook de bedoeling om, met deze tekst als leidraad, zelf de code op te stellen: het is een goede oefening op delegates en dependency properties.

9.2 Relevante namespaces

9.3 CaptionControl

Wanneer we een WPF-toepassing maken is het niet zo eenvoudig om de titelbalk bovenaan een window aan te passen (u moet de windows themes wijzigen, wat zijn effect heeft op elke toepassing). In dit hoofdstuk zullen we een eigen control maken die als titelbalk kan dienen, maar waarvan we de eigenschappen (hoogte, lettertype, gebruikte images in de controlbox, ..) volledig zelf kunnen aansturen. Het is de bedoeling een custom control te ontwikkelen, zodat de visualisatie volledig wijzigbaar is. Het voorbeeld illustreert zowel het gebruik van PARTS als STATE's.

9.3.1 PARTS

9.3.1.1 Caption control via PARTS: inleiding

Als basis voor deze tekst gebruiken we MSDN: Creating a Control That Has a Customizable Appearance.

Wanneer u een custom control maakt geeft u de visualisatie van uw control uit handen: de gebruikende programmeur (designer) is in staat om de volledige visualisatie aan te passen. Indien u rekent op de aanwezigheid van een visueel element om een correcte verwerking te kunnen garanderen (onze CaptionControl verwacht dat er 'iets' is waarmee het window kan gesloten worden) is het wenselijk om dit aan de gebruiker (en zijn ontwikkel software: Blend of Visual Studio) duidelijk te maken. Dit gebeurt op twee manieren:

- PART_ prefix (ik vind dit niet terug als guideline, en wel een Pro WPF in VB 2010, pagina 567 die stelt dat deze naamgeving sinds de introductie van VSM verouderd is);
- *TemplatePart* attributen op de customcontrol.

In de *OnApplyTemplate* methode van de custom control kunnen we controls, aangemaakt door toepassing van de template opvragen en via events koppelen aan gewenst gedrag. U moet uw code voorzien op de eventualiteit dat de gebruiker de vereiste parts niet aanbood met de gewenste naam. In dat geval is het mogelijk dat een stuk functionaliteit niet langer aanwezig is, maar uw toepassing mag hierdoor natuurlijk geen fout veroorzaken!

Figuur 9.1: Window met eigen caption

9.3.1.2 Caption Control via PARTS: uitwerking

- maak een nieuw WPF project, en plaats de WindowStyle property van het aanwezige Window op *None*. Hierdoor zal het Window bij het opstarten geen caption meer vertonen. Het is de bedoeling om nu zelf een caption control te maken die dezelfde functionaliteit aanbiedt en beter aanpasbaar is;
  - hint: u kan dit venster nog steeds afsluiten met de *Alt-F4* toetscombinatie;

Figuur 9.2: Window met WindowStyle none

- voeg via Visual Studio een *Custom Control ccCaption* toe aan het project. Hierdoor wordt een gepaste static constructor en een bijhorend theme aangemaakt. Save en compileer het project waarna u het in Blend opent;
- editeer de resource gevonden in *Generic.xaml*:
  - voorzie een grid met 4 kolommen;
  - de drie laatste kolommen worden gebruikt om de *minimize*, *maximize* en *close* functionaliteit aan te bieden. Omdat de te gebruiken images vierkant zijn zullen deze drie laatste kolommen net zo breed maken als ze hoog zijn (we binden de *Width* property aan de *ActualHeight* property van hun *Grid Ancestor*). De eerste kolom neemt de rest van de beschikbare plaats in beslag en zal de te tonen titel bevatten.
- plaats een Image in elk van de drie laatste kolommen, met als naam PART_Minimize, PART_Maximize, PART_Close;
- voorzie de custom control klasse van *TemplatePart* attributen die elk van deze drie images benoemen. Als type in het TemplatePart attribuut kiezen we niet *typeof(Image)* maar *typeof(UIElement)* om zo weinig mogelijk beperkingen aan de nodige elementen op te leggen.
  - plaats ook een control die de titel zal bevatten in de Grid, ook al zullen we deze in deze les verder niet verwerken;

- voorzie in de grid van het MainWindow een rij bovenaan waarbinnen de caption control getoond wordt. Verifieer dat de visualisatie klopt met de verwachtingen: indien u het Window resized moet ook de icoontjes resizen!
  - vergeet niet *VerticalAlignment* en *HorizontalAlignment* properties op Stretch te plaatsen en de *Stretch* property van de Images op *Fill*;
  - wellicht zal u de icoontjes die door Blend in de *Themes* folder werden geplaatst moeten verplaatsen naar een *images* folder die u zelf aanmaakt. U moet dan natuurlijk de XAML-verwijzing naar de ico-files ook aanpassen.

Om nu de gewenst acties aan de voorzien Images te koppelen gaan we als volgt te werk:

- override *OnApplyTemplate*. Gebruik *GetTemplateChild* om de UIElementen waaraan logica gekopeld moet worden (deze vermeldt in de *TemplatePart* attributen) op te halen;
  - koppel code aan het *MouseUp* event, wat overeenkomt met een *Click* event voor de gebruiker. Jammer genoeg is *Click* nog niet gekend op *UIElement*, reden waarom we *MouseUp* gebruiken.
  - om een window te sluiten moet u het window waarop de control staat kennen. U kan het window ophalen met een zeer eenvoudige recursieve routine die gebruik maakt van *VisualTreeHelper.GetParent*
    - voorzie een implementatie voor de drie bewerkingen: minimize, maximize en close.
    - u kan het Window waarbinnen een control zich bevind ook opvragen met behulp van een Window.GetWindow- oproep (met dank aan Bram Teirlynck)

Verifieer dat de gevraagde functionaliteit werkt.

9.3.1.3 Caption control via Parts: test en aanpassing

Gebruik Blend om in het formulier nieuwe template code voor de Caption control aan te maken: we vervangen minstens één van de Images door een *Button*. Merkwaardig genoeg zal die knop niet werken, ook al wordt de code doorlopen en het event gekoppeld: omdat sommige control-Types het MouseUp- event zelf afhandelen en *Handled* op *True* plaatsen wordt het event niet meer wordt doorgegeven. We zullen de event-koppeling op een andere manier moeten realiseren:

- kijk na hoe u met behulp van *AddHandler*- instructies een reeds afgehandeld event kan opvangen;
- verifieer dat het gebruik van de knop nu wel het beoogde effect heeft;
- pas ook de code voor de andere controls in de default template aan.

9.3.1.4 Caption Control via PARTS: evaluatie

- Deze manier van werken levert de client-ontwikkelaar flink wat vrijheid en geeft de control ontwikkelaar de kans om logica te koppelen aan (te verwachten aanwezige) visuele elementen;
- een nadeel is uiteraard de vereiste aanwezigheid van elementen met een specifieke naam, en het moeilijk koppelen van andere controls aan de gewenste logica (veronderstel dat we op de caption twee close- elementen wensen te plaatsen, terwijl er maar één template part hiervoor werd voorzien);
- deze methode is enkel toepasbaar indien u zelf de control ontwikkelt. Indien u een ander event dan dit dat default aan het Command is gebonden wenst op te vangen zal u gebruik moeten maken van de dependency properties benadering die wordt toegelicht in het volgende hoofdstuk.

9.4 Caption Control opgave

- maak een Caption Control zoals hiervoor beschreven en demonstreer zijn werking. Opdat u de images die u via Blend op uw controls plaats ook te zien zou krijgen gaat u best op de volgende manier te werk:
  - maak een *Images*- map in het project op hetzelfde niveau als de *themes*- map. De gemakkelijkste manier om dit te realiseren is rechts klikken in Blend en 'add folder' kiezen;
  - voeg de image- files toe aan deze directory (mbv de verkenner buiten Blend);
  - voeg deze image- files toe aan het project door in Blend rechts te klikken op de Images folder, kies *add existing item* en (multiselect!) kies de verschillende image- files. U zou deze nu ook in Blend moeten zien onder de Images folder;
  - stel deze images als source in voor de controls door deze in de keuzelijst (bij de source property) te kiezen.

met dank aan Sven Meesters (2011-2012)

9.5 States

Een visualisatie heeft hopelijk de mogelijkheid om verschillende statussen (states) van de getoonde informatie weer te geven: is een inputveld fout, teken dan een rode rand. Of zet de tekst zelf in het rood. Of plaats een uitroepteken achter het veld. Of doe een combinatie van voorgaande. Dit voorbeeld geeft aan dat de visualisatie, gekoppeld aan een bepaalde status van de gegevens, best niet bepaald wordt door het software gedeelte dat de fout detecteert, maar wel door het visualisatie gedeelte waarbinnen de fout moet benadrukt worden. We moeten ons natuurlijk niet beperken tot het al dan niet verkeerd zijn van informatie: zo zal een CheckBox het al dan niet aangevinkt zijn aangeven via een state (zie de volgende afbeelding), waarop de

visualisatie beslist hoe dit het best wordt weergegeven.

Figuur 9.3: Checkbox states visualisatie

Om dit gestructureerd mogelijk te maken zullen we als volgt te werk gaan:

- de custom control zal verschillende mogelijke states definiëren via TemplateVisualState- attributen op de control- klasse;
  - in de tekst staat beschreven hoe de onderlinge exclusiviteit gedefinieerd wordt;
  - deze states zijn trouwens ook gekend in Blend;
  - ook de .NET controls werken met dit states- systeem;

Figuur 9.4: Checkbox states via Blend

- in de XAML-code kunnen we de overgang tussen de verschillende states geanimeerd visualiseren:


```
<Grid x:Name="grd">
    <VisualStateManager.VisualStateGroups>
        <VisualStateGroup x:Name="BezetState">
            <VisualState x:Name="Bezet">
                <Storyboard x:Name="sbBezet">
                    <ColorAnimation Duration="0:0:
                </Storyboard>
            </VisualState>
            <VisualState x:Name="OnBezet">
                <Storyboard>
                    <ColorAnimation Duration="0:0:
                </Storyboard>
            </VisualState>
        </VisualStateGroup>
```

Figuur 9.5: State animaties in XAML

  - u kan deze animaties perfect via Blend aanmaken door een storyboard aan te maken en overgangen tussen states aan te geven;
  - bemerk dat de start- en doel- status niet tot de animatie behoren;
  - indien u de XAML-code zelf codeert: u moet de VisualStateManager tag onmiddellijk onder de eerste control van de visualisatie coderen.
- langs de $C^{\#}$kant van de control moeten de state- overgangen aangegeven worden. Hiervoor moet de VisualStateManager.GotoState methode worden uitgevoerd. De MSDN-topic in de link beschrijft de verschillende parameters. Ook de derde parameter heeft zijn betekenis!
  - dergelijk gedrag leunt zeer sterk aan bij behaviors die later in de cursus aan bod komen.

9.6 State opgave

Ontwikkel een verkeerslicht control volgens het PART- en State model. Het verkeerslicht wordt aangestuurd door op één van de lichten te klikken. Dit wijzigt de state van de control, waardoor een animatie dit licht ontsteekt en de andere lichten dooft (zie demo). Te volgen werkwijze:

- u *moet* werken met parts en states;
- u maakt de animatie om tussen de states te navigeren met behulp van Blend;

Figuur 9.6: State en Parte opgave: verkeerslicht

Jef Daels

Hoofdstuk 10

Behaviors

10.1 Behaviors: duiding

Door middel van dependency properties kunnen events op UIElementen worden afgehandeld. We spreken in dergelijk geval van *behaviors*. Oorspronkelijk geïntroduceerd in Blend 3.0 (en later opgenomen in .NET zelf) kan men behaviors gebruiken om een nieuw *gedrag* te koppelen aan uw UI. Indien de standaard aangeleverde behaviors niet voldoen kan u zelf een behavior ontwikkelen. In dit hoofdstuk zullen we Drag en Resize behaviors implementeren. Wellicht vind u op het internet vele gratis, interessante behaviors, naast deze die reeds in Blend worden aangeboden. Op het einde van dit hoofdstuk wordt een sectie gewijd aan het onthouden van de gewijzigde settings (afmetingen, posities, ..) in de application settings zodat bij het herstarten van de toepassing de aangepaste waardes toegepast worden. Ook een korte toelichting omtrent enumerations komt aan bod.

10.2 Relevante namespaces

- System.Windows.Interactivity

10.3 Drag behavior

Het verslepen van een UIElement is gebaseerd (en aangepast) op WPF tutorial .NET: Behaviors: we wensen de mogeljkheid te ontwikkelen om een control at run-time te verslepen. We zullen dezelfde behavior op drie manieren ontwikkelen:

- door middel van dependency properties: een goeie oefening en hopelijk verduidelijkend;
- via de Behavior die in Blend aanwezig is, maar niet bruikbaar voor Button- objecten;
- door een alternatieve Behavior te ontwikkelen, verder bouwend op de Blend *Behavior*-klasse.

10.3.1 Drag behavior: dependency property versie

We ontwikkelen een dependency property die, indien toegepast op een UIElement, toelaat dit te verslepen. Hiertoe moeten we drie events afhandelen:

- *MouseDownEvent*: we onthouden de huidige locatie (ten opzichte van de parent) en huidige muispositie en onthouden dat een *Drag*-beweging start (bMoveFlag wordt true);
- *MouseMoveEvent*: indien bMoveFlag true berekenen we de afstand die de muis aflegde sinds het MouseDownEvent en gebruiken deze afstand om de locatie van het UIElement met diezelfde afstand te wijzigen (het verplaatsen kan gebeuren door middel van een *TranslateTransform*;
- *MouseUpEvent*: we zetten de bMoveFlag op false, wat de eventuele verplaatsingen stopt.

Een beschrijving van de verschillende nodige elementen vind u in het tweede stuk van WPF tutorial .NET: Behaviors. Gelieve deze implementatie te realiseren en verifieer volgende zaken:

- de code werkt ook wanneer u een UIElement herhaalde keren (in dezelfde sessie) versleept;
- de code werkt ook voor een Button- control.

10.3.2 Drag behavior: Blend out of the box versie

In het voorgaande stukje werd een volledige eigen implementatie verzorgt, gebaseerd op dependency properties. Sinds Blend 3.0 worden een aantal behaviors standaard via Blend aangeboden, waaronder onder andere het verslepen van elementen. Het eerste stuk van hetzelfde artikel (WPF tutorial .NET: Behaviors) leert ons hoe we de Blend behavior *MouseDragElementBehavior* op een UIElement kunnen toepassen:

- volg de werkwijze van dit document en voorzie een control van de *MouseDragElementBehavior*-behavior;
- verifeer dat deze behavior niet werkt op een Button control.

10.3.3 Drag behavior: gespecialiseerde Blend versie

Omdat de standaard Blend-versie niet werkt voor een Button- control zullen we een eigen, Blend gebaseerde versie maken. We zullen hiervoor erven van *Behavior<T>* en de nodige methodes overriden. Het resultaat is een Behavior die ook vanuit Blend instelbaar is op een willekeurig UIElement. We gaan hiertoe als volgt te werk:

- de behavior werkt niet voor een Button, wellicht omdat de Button de klik- events zelf verwerkt en als handled markeert, waardoor ze niet langer *bubblen*. U kan een als *handled=true* bubbling event toch nog opvangen indien u gebruik maakt van de UIElement.AddHandler- instructie;
  - misschien is het ook wel de bedoeling van de Blend drag om niet te werken voor een Button. Immers, het draggen van een Button (mousedown, mousemove, mouseup) lijkt sterk op het klikken op de Button, en kan aanleiding geven tot verwarring;
- maak een nieuwe klasse die erft van *Behavior<UIElement>*. De *Behavior<T>* klasse wordt gevonden in een Blend libary. U kan deze library gemakkelijk terugvinden door middel van het vorige voorbeeld;
- in Behavior Generic Class wordt beschreven welke methodes overschreven kunnen worden. Eén van deze methodes is logisch sterk gelijkaardig aan de value changed handler die werd gecodeerd in de dependency property benadering. We gebruiken deze methode en copiëren de eerder ontwikkelde code in de nieuwe klasse (aangepast waar nodig);
- de nieuwe behavior kan vanuit Blend op een UIElement gesleept worden:
  - u selecteert de behavior (op basis van zijn naam) in het selectiewindow en sleept hem vanuit het selectie window op het UIElement;
  - het is niet mogelijke (in Blend 4) om de nieuwe behavior in de toolbox op te nemen, hij komt wel in de lijst van behaviors terecht.

Verifieer dat de nieuwe oplossing werkt op een Button element.

10.4 Grip behavior

Een *Grip* laat toe om een volledig window in plaats van een individuele control te verslepen. Eerder dan van scratch te herbeginnen en een bijna volledige copy te implementeren zullen we gebruik maken van overerving:

- erf van één van uw eigen drag- implementaties;
- in een volledige WPF omgeving zou het volstaan om de verondersteld aanwezige transformatie niet op het UIElement zelf toe te passen maar op het Window waarbinnen dit UIElement geplaatst werd (te vinden met de *VisualTreeHelper*). Jammer genoeg is een *Window* geen echt WPF object en kan men hierop geen RenderTransform instellen. We kunnen het window verplaatsen door zijn Top en Left properties aan te passen;

  - override de mousemove handler;
  - bereken, net zoals in de parent klasse de af te leggen afstand;
  - gebruik de *Top* en *Left* properties van het window om dit te verplaatsen;
  - het kan natuurlijk nodig zijn om een aantal elementen in de parent klasse niet langer als private maar nu als *virtual* te declareren.

10.5 Resize behavior

Een gelijkaardige (iets moeilijker) behavior is een *Resize*- behavior: het is de bedoeling om, wanneer op de rand van een control wordt geklikt, deze kan geresized worden. Hiervoor gaan we als volgt te werk:

- we erven van de Blend- Behavior classe en koppelen drie events;
- op het *MouseDownEvent* zullen we nakijken of er aan de rand wordt geklikt. Zoja wordt de cursor van het UIElement gewijzigd en onthouden we de mousedown positie;
- op het *MouseUpEvent* herzetten we de cursor tot zijn default;
- in het mousemove event verwerken we de verplaatsing op volgende wijze;
  - indien u moet resizen naar het oosten of naar onder volstaat het om de afmeting van de control aan te passen en de huidige muispositie als nieuwe mousedown te registereren;
    - indien u de mousedown positie niet vervangt wordt de afstand telkens ten opzicht van de originele positie genomen, zodat eerdere bewegingen terug geteld worden.
  - indien u naar het westen of noorden beweegt moet u de afmetingen van de control aanpassen *en* de control verplaatsen. Normaal gezien moet u de nieuwe muispositie niet registeren omdat die door het verplaatsen van de control terug samenvalt met de originele mousedown positie
    - net zoals in de Drag behavior oplossing is het verplaatsen van de control het gemakkelijkst te realiseren door middel van een translate transform.

Omdat we de *Width* en *Height* van een control manipuleren zal deze implementatie slechts werken indien deze reeds werden ingesteld. In een professionele implementatie is het nodig om, indien de Width of Height nog niet werden ingesteld, gebruik te maken van respectievelijk *ActualWidth* en *ActualHeight* om de berekeningen uit te voeren.

10.5.1 Alignment impact (Laurenz Demey 2010- 2011)

Wanneer de te resizen control *right aligned* is werkt bovenstaande code niet: de speciale afhandeling voor een resizen naar links (verbreden en verplaatsen) moet dan toegepast worden op de rechterkant (en mag niet toegepast worden op de linkerkant (analoog voor onder en bovenkant)

10.6 Behavior opgaves

- implementeer en demonstreer de Drag behavior als dependency property;
- illustreer de werking van de Drag behavior gevonden in Blend;
- implementeer en demonstreer de Drag behavior als ervend van *Behavior<T>*;
- implementeer een *Grip* behavior. Die laat toe om niet een control maar een volledig Window te verplaatsen. Illustreer de werking door deze toe te passen op de Caption controls gemaakt in Templated Controls;
- implementeer een resize behavior;
- plaats de Grip behavior op de Caption Control;
- aanrader: maak een Caption Control zoals deze in het vorige hoofdstuk, maar koppel de logica aan de UI-elementen door middel van dependency properties.
- een model examen opgave had kunnen zijn: pas de F1URL implementatie aan zodat het een behavior wordt:
  - u bent dan niet langer gebonden aan de eigen grid klasse waarbinnen de controls moeten geplaatst worden;
  - u hebt niet langer nood aan een hittest routine om te bepalen welke control de help opvraagt;
- een model examen opgave zou kunnen zijn: ontwikkel de *Behavior-* klasse zelf voor de Windows Store omgeving;

10.7 WPF en application settings

Een interessante feature in de WinForms-designer die ontbreekt in WPF-toepassingen is het gebruik van application settings (instellingen) die UI-properties over sessies heen onthouden. Mits een weinig infrastructuur werken loopt dit echter wel los:

- voeg in Visual Studio aan de properties van uw toepassing de nodige settings manueel toe (kies een geschikte naam en datatype). De designer zal een application settings aanmaken met een static default instantie (u kan de code nakijken in de file *Settings.Designer.cs*.
- koppel de te onthouden WPF eigenschappen aan hun overeenkomstige setting:

```
Height="{Binding Source={x:Static src:Properties.Settings.Default}, Path=hoogte, Mode=TwoWay}"
```

Figuur 10.1: Height en applicationsettings

- anders dan in *VB.NET* worden in *C#* de application settings niet automatisch bewaard. Daarom zullen we in het *Closed* event van het Window de applicationsettings zelf bewaren: *JDADragBehavior.Properties.Settings.Default.Save();* (waarbij JDADragBehavior de naam van het project is).

Jef Daels

10.8 Enumerations: getalwaardes en bewerkingen

In mijn eigen implementatie van de Resize behavior ontwikkelde ik en enumearation om de rand waarbij de cursor zich bevind bij te houden:

```
public enum RandPositie
    {
    Midden=0,
    Noord=1,
    Zuid=2,
    Oost=4,
    West=8
    }
```

Figuur 10.2: Enumeration definitie

Indien een variabele een combinatie van verschillende waardes moet bevatten (stel Noord en West) dan worden beide waades opgeteld met de — operator. Indien we willen testen of een waarde onder andere de Noord waarde bevat gebruiken we de *&* operator. Door de getalwaardes als machten van 2 te kiezen is elk getal slechts op exact één manier een combinatie van de voorgaande. Hierdoor is geen verwarring mogelijk.

- het getal 7 is met voorgaande definitie slechts mogelijk door combinatie van 1 (Noord), 2 (Zuid) en 3 (Oost);
- 7 & Zuid levert een waarde verschillend van 0 (midden);
- 7 & West lever 0 (midden) op.

Om de uniciteit van de combinaties te begrijpen kan een binaire invalshoek helpen:

- doordat de getallen machten van 2 zijn, zijn ze in een binaire voorstelling allemaal 1 gevolgd door een aantal nullen. Het combineren van verschillende enum-elementen zet de overeenkomstige 1-waardes actief (elke 1-waarde kan slechts door 1 enum waarde worden opgezet, gezien we geen dubbels toelaten). Het getal resultaat wordt voor het leesgemak in de code decimaal genoteerd;
- indien u de binaire voorstelling van een getal nakijkt kan u op basis van de 1-tjes nakijken welke enumwaardes deel uitmaken van dit getal. De & operator is uitermate geschikt om na te kijken of een specifieke enumwaarde (die maar één 1 bevat) deel uitmaakt van een gecombineerde set waardes.

10.9 Extra lectuur

- The Attached Behavior pattern: een theoretische duiding;
- Introduction to Attached Behaviors in WPF: een zeer eenvoudig en verklarend voorbeeld;
- WPF tutorial .NET: Behaviors: een mooi voorbeeld in twintig regels code, verifieer of het werkt bij herhaalde uitvoeringen van dezelfde sessie;
- Behavior Generic Class

Jef Daels

Hoofdstuk 11

iTextSharp

iText is een in België onwikkelde (open source) bibliotheek waarmee PDF-documenten gegenereerd en gemanipuleerd kunnen worden. Deze bibliotheek gebruik ik bijvoorbeeld om een watermerk in deze PDF te plaatsen.

11.1 Installatie

- start een nieuw WPF project;
- *Project.Manage NuGet Packages ..*;
- zoek op *iText* en installeer *iTextSharp* (de $C^{\#}$versie van iText);

11.2 PDF creatie (en opgaves)

Voor een eenvoudige tutorial omtrent het aanmaken van een nieuwe PDF verwijs ik graag naar Basic PDF Creation Using iTextSharp.

Een beetje rondzoeken op het internet helpt u wel verder omtrent een aantal topics. Zo moet u in uw test document volgende zaken realiseren:

- plaats een figuur in de pdf en herschaal deze tot 150x150;
- maak een titel paragraaf met een grotere fontsize en gecentreerd;
- maak een paragraaf die bestaat uit chunks die elk hun eigen kleur hebben;
- trek een lijn in een pdf. Let wel: het PDF coördinaten systeem is verschillend van dit dat u gewoon bent in een informatica omgeving:
  - de oorsprong is links onder op het blad, en niet links boven;
  - een grotere y- waarde ligt hoger op het blad, terwijl dit in een windows omgeving lager ligt.
- ...

11.3 Manipulatie van een bestaande PDF (en opgaves)

- maak een nieuw PDF document door een aantal PDF-documenten achter elkaar te plaatsen;
- lees een pdf document en plaats 4 logische pagina's op 1 fysisch blad;
- plaats een watermerk op elke even pagina van een document.

11.4 Tekenen (en opgaves)

Voorzie uw Sudoku oplossing van de mogelijkheid om een Sudoku te exporteren naar PDF:

Figuur 11.1: Sudoku in PDF

Let op volgende zaken:

- de opgave- getallen zijn anders gekleurd dan de oplossing- getallen;
- de 3x3- hokjes zijn van elkaar gescheiden door dikkere lijnen;

11.5 Toepassingsgebied voor iText

- *niet* als tekstverwerker;
- wel als rapport generator (veel veel sneller dan office activeX);

Jef Daels

Hoofdstuk 12

SQLServer en .NET integratie

Relationele database laten meestal toe om logica in de vorm van stored procedures in de database op te nemen. Naast administratieve voordelen betreffende security en onderhoud is ook de potentiële performance winst een troef: berekeningen op de server dienen niet alle nodige gegevens over het netwerk naar de client te sturen, enkel het resultaat sturen is voldoende. Oorspronkelijk diende deze logica in een RDBMS specifieke, behoorlijk ouderwetse taal gedefinieerd te worden. Sinds enige jaren bestaat de mogelijkheid om de logica in uw favoriete programmeeromgeving te ontwikkelen en deze daarna in de server te plaatsen. Dit hoofdstuk behandelt het opnemen van .NET programmatuur in SQLServer.

12.1 Nodige softwares

- Visual Studio 2012;
- SQL Server 2012;
- SQL Server Datatools 2012.

Installatie van de SQL Server Datatools maakt een nieuwe Visual Studio versie beschikbaar. Deze dient u te gebruiken voor deze les.

12.2 Kwadraten berekenen

Wellicht weinig zinvol in een database context, maar gemakkelijk om het principe uit te leggen is het aanmaken van een functie die bij oproep een tabel van kwadraten aanlevert. Indien u zelf meer voorbeelden wil zal Usage Scenarios and Examples for Common Language Runtime (CLR) Integration wellicht een goed startpunt zijn.

- start *SQL Server Data Tools 2012*, een *Visual Studio* oplossing;
- voeg aan de SQL Server object explorer uw lokale SQL Server 2012 installatie toe;
- voeg de database SQLNET toe. Dit is de database waarbinnen we onze test zullen uitvoeren;
- lees mij;
- *Create New Project* op de zonet gemaakte database, met als naam: *KwadraatDemo*;
- voeg de klasse *Kwadraat* toe:

```
public class Kwadraat
    {
    private SqlInt64 _getal, _kwadraat;
    public Kwadraat(SqlInt64 getal, SqlInt64 kwadraat)
        {
        _getal = getal; _kwadraat = kwadraat;
        }
    [SqlFunction(FillRowMethodName = "VentileerRijen", TableDefinition = "getal bigint, kwadraat bigint")]
    public static IEnumerable MaakKwadraten(int maxgetal)
        {
        try
            {
            ArrayList kwadraten = new ArrayList();
            for (int i=1;i<=maxgetal;i++)
                kwadraten.Add(new Kwadraat(i, i*i));

            return kwadraten;
            }
        catch (Exception ex) { return null; }
        }
    public static void VentileerRijen(object okwad, out SqlInt64 getal, out SqlInt64 kwadraat)
        {
        Kwadraat kwad = okwad as Kwadraat;
        getal = kwad._getal;
        kwadraat = kwad._kwadraat;
        }
```

Figuur 12.1: Kwadraat code

- compileer en publish deze code;
  - voeg de nodige referenties aan het project toe;
  - bemerk dat SQLServer datatypes verschillend zijn van .NET datatypes;

  - bij het publishen moet u de nodige parameters voorzien om in de juiste database, op de juiste server te deployen;
- indien het publishen lukt zal u in de database onder de *table value functions* de routine *MaakKwadraten* terug vinden.
- u kan deze functie gebruiken om een tabel kwadraten te selecteren:

```
use SQLNET
 select * from MaakKwadraten(10);
```

100 %

Results | Messages

	getal	kwadraat
1	1	1
2	2	4
3	3	9
4	4	16
5	5	25
6	6	36
7	7	49
8	8	64
9	9	81
10	10	100

Figuur 12.2: Kwadraten selecteren

Omtrent bovenstaande code verwijs ik graag naar het internet voor een eerste toetsting van uw vragen.

12.3 SQL Server tabellen raadplegen

Wanneer u in een klassieke toepassing SQL Server tabellen raadpleegt dient u een database connectie te configureren. Omdat we nu code ontwikkelen die in SQL Server opgenomen zijn en dus geconsulteerd worden via een bestaande connectie die we willen hergebruiken zal dit op een net iets andere manier gerealiseerd worden

- maak een tabel (vergeet de PK niet) in de SQLNET database en voorzie deze van een aantal rijen;
- lees mij;
- gebruik uw verzamelde kennis (deze les en vorige jaren) om een TVF te maken die iets uit die tabel ophaalt en terug geeft. Maak bijvoorbeeld een select met een where- conditie waarbij een parameter wordt meegegeven (zie demo).

De kans is groot dat dit niet onmiddellijk lukt, wat ons onmiddellijk de nood aan debugging demonstreert.

12.4 Debugging SQLServer

12.4.1 Debugging startend vanuit het Visual Studio project

Het opzetten van de debugging in SQLServer 2012 gebeurt net iets anders dan in de vorige SQL Server versies:

- in de *SQL Server Object Explorer* (dit is *niet* SQL Server Explorer) kan u door te rechtsklikken het debuggen aanzetten;
- in de properties van het project verifieert u in de *Debug* farde of de connection settings in orde zijn. Gebruik de edit- knop indien dit niet het geval zou zijn;
- voeg een script toe aan het project waarin u het te testen SQL- statement opneemt. Opnieuw in de debug farde stelt u het startup script in op dit script (er staat een browse knop helemaal rechts);
- plaats break points in uw .NET code en start het project. Indien alles goed verloopt komt u in een debug sessie terecht.

Het debuggen, interpreteren en oplossen is voor de lezer.

12.4.2 Debugging van een SQL Server call gestart buiten het project

Indien bijvoorbeeld in Management Studio software een query gestart wordt die code uitvoert waarvan wij het project hebben kunnen we deze uitvoering als volgt debuggen:

- open het Visual Studio project en attach aan *sqlservr.exe*;
  - *Tools.Attach to process ...*;
  - zet *show processes from all users* op omdat de sqlserver service wellicht niet met uw eigen userid gestart is;
  - bij het attachen krijgt u een beveiligings boodschap die u waarschuwt voor eventuele risico's;
  - dit attachen brengt uw project trouwens in een running state;
- voer een select statement uit die de code in uw Visual Studio project uitvoert. Indien u een breakpoint plaatste zal de uitvoering daar onderbroken worden en kan u uw gebruikelijke debug acties starten.

12.5 Theorie vragen

Met onderstaande vragen test u uw begrijpen van de software en krijgt u een idee omtrent de mogelijke detailvragen voor dit hoofdstuk.

- het SqlFunction- attribuut heeft een FillRowMethodName parameter. Deze wordt in onze klasse niet opgeroepen. Leg uit wat de functie van deze methode is.
- het SqlFunction- attribuut heeft een TableDefinition parameter. Welke zijn de mogelijke datatypes?
- de datatypes gebruikt in de TableDefinition parameter zijn niet identiek aan deze gebruikt in de FillRowMethode. Verklaar!
- opdat .NET programma's in SQLServer bruikbaar zouden zijn moet een setting worden ingesteld. Is deze setting een server, database of assembly instelling?
- opdat uw .NET programma's in SQLServer via Visual Studio debugbaar zouden zijn moet een setting worden ingesteld. Is deze setting een server, database of assembly instelling?

Jef Daels

Hoofdstuk 13

Windows installer oplossingen

Om een desktop toepassing te verspreiden is het wenselijk om gebruik te maken van een MSI- bestand (Store toepassingen worden verspreid via de Store). Dergelijk installatiebestand ondersteunt een aantal standaard-acties zoals daar zijn: het voorzien van desktop- en start-icoontjes, het voorzien van een *uninstall* mogelijkheid, het automatisch mee- installeren van de nodige dll's en dergelijke meer.

Vanaf versie 2012 van Visual Studio verdwijnt de mogelijkheid om *Visual Studio setup-projecten* in een solution op te nemen. Deze feature werd vervangen door de introductie van *InstallShield Limited Edition for Visual Studio*, de gratis variant van de klassieker InstallShield. Het is belangrijk op te merken dat er ook andere al dan niet gratis softwares bestaan waarmee *MSI* pakketten kunnen ontwikkeld worden. In deze cursus komen het reeds vermeldde *InstallShield Limited Edition* en WiX (ontwikkeld door Microsoft) aan bod.

13.1 Demo programma

Ontwikkel een WPF-toepassing met volgende features:

- er worden twee assemblies aangeleverd: een .exe en een .dll;
- de UI van de .exe toont een knop die, wanneer hij wordt geklikt, een string uit de .dll ophaalt en toont in een MessageBox.
- we hebben een .ico file die we als desktop icoon wensen te gebruiken voor deze toepassing.

13.2 InstallShield Limited Edition

13.2.1 Download en installatie

Even Googlen en u vindt een gepaste download. Het resultaat van de installatie integreert netjes met Visual Studio waar een nieuw project type beschikbaar wordt gesteld onder *Other Project Types/Setup And Deployment*.

13.2.2 Aanmaken van een MSI-pakket

Het is de bedoeling dat de student zelf wat experimenteert met de software. U dient minstens volgende zaken te realiseren:

- maak een setup waarin minstens twee files voorkomen die geïnstalleerd moeten worden;
- wijzig de schermen van het installatie proces: gebruik images en wijzig de licentie tekst;
- de installatie moet een desktop-icon en een startmenu icon voorzien (het startmenu icon werkt momenteel niet onder windows 8).

13.2.3 Redistributables

Wellicht is uw software afhankelijk van andere softwares, zoals het .NET framework, SQLServer, .. . U kan deze softwares meenemen in het installatiepakket. Om één of andere reden lukt dit me niet voor het .NET framework bij het gebruik van de gratis InstallShield.

Jef Daels

13.2.4 Evaluatie InstallShield Limited Edition

Ik ben er niet wild van:

- na één jaar Windows 8 is het geïnstalleerde programma nog steeds niet op het start scherm te plaatsen;
- redistributables opnemen in het pakket (of deze automatisch door de client laten downloaden) lukt me niet. Misschien enkel in de betalende versies?
- een groot aantal features wordt in de gratis versie enkel vermeld maar kan pas gebruikt worden in de betalende versies:
  - het bepalen van dependencies gebeurt niet in de gratis versie;
  - setup- design;
  - ...

Deze software lijkt me bruikbaar in eenvoudige situaties, zoals een lescontext. Een voordeel is dan weer wel dat de wizard driven interface gemakkelijk in gebruik is. Wellicht is de betalende versie ook bruikbaar in complexere situaties.

13.3 WiX

Wix is een XML gestuurde software met een stijle leercurve. Een tutorial is dan wellicht ook op zijn plaats.

13.3.1 Download en installatie

Google .. .

13.3.2 Aanmaken van een MSI-pakket

Het is de bedoeling dat de student zelf wat experimenteert met de software. U dient minstens volgende zaken te realiseren:

- maak een setup waarin minstens twee files voorkomen die geïnstalleerd moeten worden;
- wijzig de schermen van het installatie proces: gebruik images en wijzig de licentie tekst;
- de installatie moet een desktop-icon en een startmenu icon voorzien (het startmenu icon werkt momenteel niet onder windows 8).

13.3.3 Redistributables

Wellicht is uw software afhankelijk van andere softwares, zoals het .NET framework, SQLServer, .. . U kan deze softwares meenemen in het installatiepakket. In een Wix- benadering moet u een extra *Bootstrapper Project* (dit is een speciaal project type) aanmaken. Dit project type specifieert een *Chain* van te installeren softwares. Uw eigen software (via het WiX installatie project) en de te installeren software (bijvoorbeeld het .NET framework) maken hier onderdeel van uit.

13.3.4 Evaluatie WiX

De leercurve van WiX is *veel* stijler dan deze van InstallShield. Toch verkies ik Wix boven InstallShield (een privé mening), wellicht om volgende redenen:

- de gratis versie is geen beperkte versie;
- de (complexe) XML- aansturing (en de online documentatie) geven me toch wel het idee dat ik de meeste situaties geconfigureerd kan krijgen, waar ik met de gratis InstallShield versie toch wel wat beperkingen wordt opgelegd.

13.4 Welke installer?

13.4.1 Vergelijking InstallShield en Wix

Indien geld geen probleem zou zijn verwacht ik een grote meerwaarde bij het gebruik van een betaalde InstallShield installatie. Indien enkel de gratis versies in overweging worden genomen verkies ik WiX omdat ik vrees dat de free InstallShield versie niet voor alles toereikend zal zijn, en ik op termijn dus toch WiX zal nodig hebben.

13.4.2 Vergeet niet:

Er zijn nog andere installatie softwares, zowel gratis als betalend, dit hoofdstuk heeft geenszins de bedoeling om een uitputtende vergelijking tussen de verschillende mogelijkheden te maken.

13.5 Voorbeeld labo examen opgave

- maak een setup voor uw sudoku programma, zowel met InstallShield als Wix;
- voorzie in de setup ook een PDF met een getekende Sudoko die mee geïnstalleerd wordt (de veronderstelde help).

Jef Daels

Hoofdstuk 14

AOP: Aspect Oriented programming

14.1 AOP duiding

Bij het ons welbekende programmeren is het gebruikelijk om programmalogica logisch op te delen in verschillende klasses en methodes: elke methode staat in voor de correcte afhandeling van een deelprobleem. Hoewel dit suggereert dat elk probleem op één welbepaalde plaats wordt afgehandeld zijn er *aspecten* van het programma die zich *niet* tot een welbepaalde plaats beperken: *logging*, *parameter validatie*, *authorisatie checks*, ... komen telkens opnieuw, op verschillende plaatsen terug. *Aspect Oriented Programming* biedt de mogelijkheid om de aspect gerelateerde code op één welbepaalde plaats te coderen:

.NET ondersteunt AOP door middel van de ContextBoundObject- klasse. Omwille van de beperktheid van deze oplossing zullen we voor deze lessen gebruik maken van *PostSharp*, hoewel dit natuurlijk niet het enige alternatief is. In Rating of Open Source AOP Frameworks in .NET worden een aantal frameworks met elkaar vergeleken. Recent werd *PostSharp* een commercieel produkt (het was oorspronkelijk gratis), reden waarom we gebruik maken van een trial versie.

Onderstaande afbeelding illustreert het beperken van een examenpunt tot een getal tussen nul en twintig, door middel van een attribuut. Het afdwingen van de parameter- beperking zal gebeuren door middel van het *OnEntry*- aspect.

```
public static void ingaveStudentModulePunt(string student, string module,[AOPIntegerRange(0,20)] int punt)
    {
    ////parameters testen
    //if (punt < 0) throw new Exception(punt + " is kleiner dan 0");
    //if (punt > 20) throw new Exception(punt + " is groter dan 20");

    Console.WriteLine(student + " haalde voor " + module + " " + punt + " punten.");
    }
```

Figuur 14.1: AOP: parameter annotatie

14.1.1 Relevante namespaces

- PostSharp.

14.2 PostSharp

14.2.1 PostSharp kennismaking: methode-entry teller

Indien u versie 2.5 installeert (Google) gaat u na installatie als volgt te werk:

- ontwikkel een Fibonacci oplossing
- use *Project.Manage NuGet packages* to add the *Postsharp-Aspects* package;
- maak een eigen klasse *clsEntryTeller* die erft van *PostSharp.Aspects.OnMethodBoundaryAspect*;
- override de *OnEntry* methode. Deze wordt uitgevoerd telkens een methode, voorzien van het *clsEntryTeller* attribuut wordt gestart (we zullen dit attribuut op alle methodes van een klasse toepassen door het attribuut op klasse niveau te definiëren);
- voorzie logica om de uitvoering van deze methode te registreren. Dit zal wellicht gebeuren door middel van een static field in *clsEntryTeller* dat voor elke methode het aantal uitvoering bijhoudt (gebruik een *Dictionary*?);
- voorzie logica (statisch?) om de informatie, verzameld in *clsEntryTeller*, in de console te dumpen.
- programmeer wat test- logica en voorzie de te tellen methodes van het *clsEntryTeller* attribuut. Dit kan gebeuren op verschillende manieren:
  - u definieert het attribuut voor de methodehoofding (bruikbaar voor een klein aantal methodehoofdingen);
  - u definieert het attribuut op klasse (of assembly)- niveau en vermeld (met behulp van reguliere expressies) welke methodes al dan niet van het attribuut moeten voorzien worden.
- test het resultaat.

PostSharp weeft IL-code rond de methodeoproepen waarop het *OnMethodBoundaryAspect* werd toegepast. De door PostSharp geïntroduceerde namen bevatten tekens die door ons bij het programmeren niet gebruikt kunnen worden. Dit vermijdt potentiële naamconflicten tussen onze code en de gegenereerde code. Zoals reeds werd vermeld in het hoofdstuk omtrent runtime compilatie wordt dit code-weven gestart door in het *MSBuild* proces in te grijpen.

Jef Daels

ILSpy maakt duidelijk hoe de uiteindelijke gegenereerde IL code er uit ziet (voor de hieronder gegenereerde IL-code werd ook de *OnException* methode overriden, zodat het hier getoonde try-catch blok werd gegenereerd):

```
public class Wiskunde
{
    [System.Diagnostics.DebuggerNonUserCode, System.Runtime.CompilerServices.CompilerGenerated]
    internal sealed class <>z__Aspects{...}
    public static int Fibonacci(int i)
    {
        MethodExecutionArgs methodExecutionArgs = new MethodExecutionArgs(null, null);
        MethodExecutionArgs arg_13_0 = methodExecutionArgs;
        System.Reflection.MethodBase m = Wiskunde.<>z__Aspects.m3;
        arg_13_0.Method = m;
        Wiskunde.<>z__Aspects.a0.OnEntry(methodExecutionArgs);
        int result;
        try
        {
            int num;
            if (i <= 0)
            {
                num = 0;
            }
            else
            {
                if (i == 1)
                {
                    num = 1;
                }
                else
                {
                    num = Wiskunde.Fibonacci(i - 1) + Wiskunde.Fibonacci(i - 2);
                }
            }
            result = num;
        }
        catch (System.Exception CS$0$5__exception_58)
        {
            Wiskunde.<>z__Aspects.a0.OnException(methodExecutionArgs);
            throw;
        }
```

Figuur 14.2: PostSharp en ILSpy

14.2.2 PostSharp en parameter annotaties

Het codevoorbeeld in de duiding illustreert de opzet: we wensen *précondities* aan formele parameters op te leggen door middel van attributen. In het hieronder uitgewerkte voorbeeld zal een *préconditie* enkel van toepassing zijn op een individuele parameter. We wensen de volgende situtatie te bekomen:

- bij de start van een methode zal het *OnEntry* aspect de eventueel aanwezige parameter-attributen (te vinden in de formele methode declaratie) checken;
- elk préconditie attribuut verifieert of de actuele parameter waarde voldoet. Indien niet wordt er een fout gegooid;
  - we leggen aan elk préconditie attribuut de interface *IAOPValidatie* op. Op deze manier is het gemakkelijk na te gaan of een parameter attribuut door onze infrastructuur moet gechecked worden;
  - we voorzien die interface van een *valideer* methode. Op die manier kan elk attribuut op dezelfde manier verwerkt worden (zodat we met een eenvoudige lus alle attributen kunnen overlopen en verwerken).

Meer in detail gaan we voor de attributen als volgt te werk

- definieer de interface *IAOPValideer* en voorzie de *valideer*- methode. Het return type is *void* en de routine heeft één parameter van type *object* (op deze manier kan het valideren van een datum of string door dezelfde virtuele methode opgevangen worden;
- definieer préconditie klasses die deze interfaces implementeren. De constructor (gebruikt in de attribuut definitie van de formele parameter) neemt welilcht één of meerdere parameters (range- condities nemen wellicht twee argumenten in de constructor);
- implementeer de *valideer*- methode. Deze throwt een *AOPValidatieException* indien de voorwaarde niet voldaan is.

Voor het aspectorientatie gedeelte voorzien we volgende code:

- voorzie een (*PostSharp*) *OnMethodBoundaryAspect*;
- override de *OnEntry*- methode. Via *args.Method.GetParameters* bekomt u de formele parameters van de oproep (*args* bevat de actuele parameters). De formele en actuele parameters stemmen positioneel met elkaar overeen (de eerste actuele parameter (postie 0) stemt overeen met de eerste formele parameter (ook positie 0)).
- schrijf wat code om het voldaan zijn van de eventuele attributen te checken (twee lussen, vijf lijntjes logica code (dus zonder accolades of routine hoofdingen))!

Houd ook rekening met volgende zaken:

- wanneer u uw code test, beperk u dan niet tot précondities die niet voldaan zijn. Indien de préconditie wel voldaan is moet het programma uiteraard verder werken;
- plaats eens meerdere précondities op hetzelfde argument;
- in de *PostSharp* documentatie vinden we dat het construeren van de method- informatie een inefficient proces is. De hierboven geschetste implementatie zal deze informatie bij elke methode oproep verzamelen. U kan dit beperken door de methode informatie te verzamelen in de *RuntimeInitialize* uitvoering, en deze dan op module niveau (voor deze specifieke methode, dus niet *statisch*) te bewaren.

Hopelijk realiseert u zich dat de attributen die worden geplaatst bij de formele parameter definities op zich geen AOP introduceren: we plaatsen met deze attributen meta-informatie op de parameters, die dan door de AOP-verwerking (de method- entry) gebruikt worden om de extra logica (de aspect logica) te sturen.

14.2.3 PostSharp en performance counters

Performance Counters zijn (sinds NT) *de* manier om prestatie cijfers omtrent uw toepassing aan het operating systeem mede te delen. Third party toepassingen kunnen deze counters lezen en visualiseren (of rapporteren):

Figuur 14.3: PDFStamper performance counter

Bovenstaande figuur toont de performancemonitor resultaten van mijn PDFWatermark toepassing (op elke bladzijde wordt een watermerk geplaatst). U merkt duidelijk dat er twee uitvoeringen van het programma voorkwamen. De hier getoonde performance monitor zit standaard in Windows (de screenshot is genomen in een Vista- installatie).

Windows en .NET leveren reeds een grote lijst performance counters. U kan deze gemakkelijk nakijken vanuit de *Server Explorer* in *Visual Studio*:

Figuur 14.4: Performance counters in Visual Studio

Bij een eerste test zal u wellicht geïnteresseerd zijn in hardware gerelateerde counters zoals de *batterijstatus* of *processor* gegevens.

14.2.3.1 Eigen performance counters

Voor het publiceren van eigen performance counters verwijs ik naar How do I... Publish a custom application performance counter in *C#*?. U zal dit uiteraard moeten beheersen vooraleer u uw eigen performance counters via AOP kan aansturen (het PerformanceCounterType is zeer belangrijk).

14.2.3.2 AOP en performance counters

De performance counters worden vooral gebruikt om toepassingen te profileren: hoe gedraagt een toepassing zich tijdens haar uitvoering? Performance counters kunnen dan ook best gebruikt worden om betekenisvolle activiteiten van uw toepassing te publiceren. Denken we hierbij bijvoorbeeld aan: het aantal vewerkte klanten (per seconde), het aantal failures (per seconde), .. . Om dit met AOP te ondersteunen denken we aan volgende implementatie:

- we wensen een performance attribuut op methode niveau toe te passen, en de mogelijkheid te hebben om op *OnEntry*, *OnException* en *OnSuccess* in te grijpen. *OnMethodBoundaryAspect* is hiervoor uitermate geschikt;
- op basis van de gegevens, nodig om een goede performance counter te definiëren, zullen we bij het overerven extra parameters aan het attribuut toevoegen;
- gebruik deze extra info om op de gewenste plaatsen een performance counter aan te sturen: u bent natuurlijk *niet* verplicht om zowel het falen als slagen van een methode uitvoering te registreren.

14.3 AOP oefeningen

- definieer een attribuut, plaatsbaar op een formele parameter, dat gecombineerd met AOP (ook zelf te maken) verifieert dat een integer waarde zich tussen een onder- en bovengrens bevindt;
- definieer een attribuut, plaatsbaar op een formele parameter, dat gecombineerd met AOP (ook zelf te maken) verifieert dat een geboortedatum een reeds geboren persoon (datum niet in de toekomst) van maximaal 120 jaar definieert;
- definieer een AOP-attribuut, plaatsbaar op een methode, waarmee wordt afgedwongen dat de uitvoerder van een routine een administrator op het toestel moet zijn;
- definieer een performance counter die met behulp van AOP op methodes kan worden geplaatst. De counter telt het aantal uitvoeringen van een routine. Gebruik deze om het aantal uitvoeringen van een aantal van de recursieve methode oproepen uit de voorbije lessen te tellen:
  - tel het aantal recursieve oproepen in uw fibonacci- oefening;
  - tel het aantal recursieve oproepen in uw backtrack- oefeningen (N-Queens en Sudoku)
  - tel het aantal PDF pagina's dat geschreven wordt met uw PDF- merge oefening.

  Het resultaat van deze performance counters moet in de performance monitor zichtbaar zijn

Jef Daels

Hoofdstuk 15

Query Notification

Query Notification biedt de mogelijkheid om gegevens wijzigingen in de database via een event in uw client programa op te vangen. Deze techniek vooral zinvol wanneer u weinig veranderende gegevens cacht in uw client (het bijhouden van lijsten informatie).

Het opzetten van *Query Notifications* is niet zo moeilijk maar vergt enig lees- en opzet werk.

15.1 Query notification oefening

Demonstreer het kunnen werken met Query Notification.

Jef Daels

Hoofdstuk 16

Discussie topics

16.1 Duiding

Als ontwikkelaar moet u in team kunnen werken. In dit team zijn wellicht ook andere ontwikkelaars aanwezig. De hieronder beschreven discussie topics worden door de klas besproken tot een consensus omtrent oplossing wordt bekomen:

- er wordt slechts één algoritme per klas per topic aanvaard;
- iedereen implementeert dit algoritme tegen de volgende les (absentie systeem);
- de week erna volgt een nabespreking, die wellicht resulteert in een herdiscussie;
- de begeleidende docent probeert de discussie *niet* te sturen.

Naast de oplossing voor de technische problemen zijn vooral volgende elementen van belang:

- ook al bent u overtuigd van het werken van uw oplossing, toch moet u ook de oplossing van iemand anders kunnen aanhoren:
  - welke punten uit zijn oplossing zijn een verbetering voor mijn eigen oplossing?
  - welke punten uit zijn oplossing zijn minder goed dan mijn eigen oplossing, en hoe kan ik dit beargumenteren
- in een werksituatie is het niet genoeg om gelijk te hebben: u moet de collega's overtuigen van uw gelijk (vermijd het Cassandra syndroom)
  - hoe krijg ik een complexe redenering uitgelegd? gebruik ik voorbeelden? terminologie?
  - hoe beschrijf ik het foutieve in een redenering?

16.2 Sinus berekenen mbv Taylor reeks

Het berekenen van een sinus kan gebeuren door middel van een Taylor reeks. U vindt een beschrijving hiervan op Wikipedia (Taylor reeks).

Groepsdiscussie: bespreek de te implementeren oplossing.

Jef Daels

Hoofdstuk 17

Global Assembly Cache: GAC

17.1 Global Assembly Cache: duiding

.NET herintroduceerde *copy deployment*, reeds bekend vanuit het *DOS*-tijdperk. Copy deployment bestaat erin dat de installatie directory van de ene naar de andere computer wordt gecopieerd, waarna dit programma ook op de tweede computer werkt.

In een tijd toen disk-space duur was (het DOS tijdperk) introduceerde dit veel overtollige copiën van dezelfde, door iedereen gebruikte dll's. Ook is het niet altijd gemakkelijk voor een programma om het reeds geïnstalleerd zijn van een ander programma te achterhalen: in theorie moet je de ganse schijf doorzoeken. Om aan deze nadelen tegemoet te komen ontwikkelde Microsoft de *registry*: elk programma dat zich installeerde registreerde zichzelf en zijn onderdelen in de registry. Nieuwe versies werden over bestaande versies heen gecopieerd (hun locatie was te vinden in de registry), en programma's vonden elkaar terug door de registry te ondervragen.

Dit gebruik van de registry resulteerde in tal van problemen, in detail beschreven in DLL hell(Wikipedia):

- indien een nieuwe versie niet 100% backwards compatibel is zal een programma dat vroeger werkte dit potentieel niet meer doen: het is perfect mogelijk om twee programma's te ontwikkelen die niet op hetzelfde toestel geïnstalleerd kunnen worden;
- het installeren van twee versies van hetzelfde programma is quasi onmogelijk;

.NET herintroduceert copy-deployment: elke toepassing komt met zijn eigen set dll's in zijn installatie directory: er is geen versie conflict meer. Voor systeem routines is dit een moeilijk houdbare zaak: sommige dll's zullen we toch centraal voor alle toepassingen wensen te installeren. Dit gebeurt in de Global Assembly Cache.

17.1.1 Relevante namespaces

17.2 GAC: technische duiding

puter ▸ Local Disk (C:) ▸ Windows ▸ assembly ▸

Assembly Name	Version	Cul...	Public Key Token	Proces...
Expression.DevHost	3.0.0.40...		31bf3856ad364e35	x86
Expression.DevHost.resources	3.0.0.40...	en	31bf3856ad364e35	x86
Extensibility	7.0.330...		b03f5f7f11d50a3a	
GAC2008	8.0.0.0		dcc8364b2810e1dc	MSIL
GAC2008	7.0.0.0		dcc8364b2810e1dc	MSIL
GAC2008	5.0.0.0		dcc8364b2810e1dc	MSIL
GAC2008	1.0.0.0		dcc8364b2810e1dc	MSIL
IACore	1.7.622...		31bf3856ad364e35	
IALoader	1.7.622...		31bf3856ad364e35	
IEExecRemote	2.0.0.0		b03f5f7f11d50a3a	MSIL
IEHost	2.0.0.0		b03f5f7f11d50a3a	MSIL
IEHost	2.0.0.0		b03f5f7f11d50a3a	MSIL
Interop.Dso	5.2.0.0		89845dcd8080cc91	MSIL

Figuur 17.1: Global Assembly Cache (iets oudere versie, Win 7)

Microsoft.Web.Administration.resources	7.9.0.0	es	31bf3856ad364e...	MSIL
Microsoft.Web.Administration.resources	7.9.0.0	de	31bf3856ad364e...	MSIL
Microsoft.Web.Administration.resources	7.9.0.0	cs	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation	7.1.0.0		31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	zh-CHT	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	zh-CHS	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	tr	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	ru	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	pt	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	pl	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	ko	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	ja	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	it	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	fr	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	es	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	en	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	de	31bf3856ad364e...	MSIL

Figuur 17.2: Global Assembly Cache (iets oudere versie, Win 8)

- er komen assemblies met identieke namen voor (dit is onmogelijk voor files in gewone directories). Dit maakt het mogelijk om in de GAC aan versie beheer te doen (dit voorkomt de problemen die optreden wanneer incompatibele versies elkaar overschrijven);
- de assemblies hebben een versie nummer. Hierdoor wordt het mogelijk om een programma te koppelen aan een specifieke versie van een assembly;

Niet zichtbaar op de screenshots, maar toch relevant:

- een assembly in de GAC *moet strong named* zijn;
- bij installatie van een nieuwe versie van een assembly kan met een *policy*- file opgegeven worden welke versies vervangen worden: het *redirecten* van assemblies. Deze *policy* files worden ook in de *GAC* geplaatst, en moet dan ook strongly named zijn. Op deze manier kan een upgrade automatisch geactiveerd worden voor programma's die een verouderde assembly gebruiken;
- een programma config file kan dit redirecten overschrijven, en verwijzen naar nog een andere (of de oorspronkelijke) assembly. De config file van het programma heeft voorrang op de assembly policy file.

  Het is belangrijk dat de programma config file voorrang heeft op de assembly policy file: indien een nieuwe versie onverhoopt toch problemen oplevert, kan men per individueel programma beslissen met welke assembly versie men blijft verder werken.
- vanaf Visual Studio 2010 is de GAC meer folder georiënteerd: C:\Windows\Microsoft.NET\assembly\GAC_MSIL\GACDLL\v4.0_1.0.0.0__72e3a55536524714

Figuur 17.3: Eigen GAC-dll's, Win 8)

17.3 Ontwikkelen voor en met de GAC

17.3.1 Strong named assemblies

Het MSDN artikel omtrent Strong-named assemblies beschrijft duidelijk *het waarom* omtrent named assemblies:

Strong-Named Assemblies

msdn

A strong name consists of the assembly's identity — its simple text name, version number, and culture information (if provided) — plus a public key and a digital signature. It is generated from an assembly file (the file that contains the assembly manifest, which in turn contains the names and hashes of all the files that make up the assembly), using the corresponding private key. Microsoft® Visual Studio® .NET and other development tools provided in the .NET Framework SDK can assign strong names to an assembly. Assemblies with the same strong name are expected to be identical.

You can ensure that a name is globally unique by signing an assembly with a strong name. In particular, strong names satisfy the following requirements:

- Strong names guarantee name uniqueness by relying on unique key pairs. No one can generate the same assembly name that you can, because an assembly generated with one private key has a different name than an assembly generated with another private key.
- Strong names protect the version lineage of an assembly. A strong name can ensure that no one can produce a subsequent version of your assembly. Users can be sure that a version of the assembly they are loading comes from the same publisher that created the version the application was built with.
- Strong names provide a strong integrity check. Passing the .NET Framework security checks guarantees that the contents of the assembly have not been changed since it was built. Note, however, that strong names in and of themselves do not imply a level of trust like that provided, for example, by a digital signature and supporting certificate.

When you reference a strong-named assembly, you expect to get certain benefits, such as versioning and naming protection. If the strong-named assembly then references an assembly with a simple name, which does not have these benefits, you lose the benefits you would derive from using a strong-named assembly and revert to DLL conflicts. Therefore, strong-named assemblies can only reference other strong-named assemblies.

See Also

Global Assembly Cache | Signing an Assembly with a Strong Name

Figuur 17.4: Strong-named assemblies (MSDN)

Om zelf een strong-named assembly te creeëren gaan we als volgt te werk:

- maak een nieuw project aan (voor dit voorbeeld een class library, hoewel natuurlijk ook alle andere types strong-named kunnen zijn);
- via de project properties in Visual Studio, in de signing farde kiest u:
  - *sign the assembly*;
  - kies een bestaande *strong name key file*, of maak een nieuwe aan. Hierdoor wordt aan het project een *snk*- file toegevoegd. Wanneer u het project herbuild zal de resultaat assembly strong-named zijn.
    - via *sn.exe*- commando kan u ook zelf een key-pair file aanmaken (zie ook How to: Create a Public/Private Key Pair.

- omdat we deze DLL later zullen gebruiken om assembly redirecting te testen zullen we een klasse voorzien die het versie nummer van de assembly teruggeeft.

  System.Reflection.Assembly.GetExecutingAssembly().GetName().Version is een startpunt om de major, minor en build gegevens van een assembly op te vragen.

17.3.2 Gacutil.exe

Dit commando laat toe om de GAC te manipuleren en ondervragen. MSDN beschrijft de parameterlijst, even googlen brengt u op de internetsnelweg.

Indien u een niet *strong-named* dll in de GAC probeert te plaatsen krijgt u het volgende resultaat:

Figuur 17.5: Gacutil -i Strong-named failure

Wanneer de assembly wel strong-named is:

Figuur 17.6: Gacutil -i Strong-named success

U kan de aanwezigheid in de cache verifëren:

Figuur 17.7: Gacutil -l

17.3.3 DLL-versies maken

Een assembly versie bestaat uit major, minor, build en revision getallen. U kan deze instellen via de project properties (application, assembly information):

Figuur 17.8: Assembly versies

U kan de versie van een file opvragen via zijn properties:

Figuur 17.9: File properties

Het is perfect mogelijk om ook deze DLL in de GAC te plaatsen. Op dat moment bevat de GAC twee dll's van dezelfde assembly.

17.3.4 Redirecting assemblies

17.3.4.1 Welke assembly wordt geladen?

Het zoeken naar assemblies wordt beschreven in How the Runtime Locates Assemblies. Hier vindt u onder andere:

The runtime uses the following steps to resolve an assembly reference:

1. Determines the correct assembly version by examining applicable configuration files, including the application configuration file, publisher policy file, and machine configuration file. If the configuration file is located on a remote machine, the runtime must locate and download the application configuration file first.
2. Checks whether the assembly name has been bound to before and, if so, uses the previously loaded assembly.
3. Checks the global assembly cache. If the assembly is found there, the runtime uses this assembly.
4. Probes for the assembly using the following steps:
   a. If configuration and publisher policy do not affect the original reference and if the bind request was created using the Assembly.LoadFrom method, the runtime checks for location hints.
   b. If a codebase is found in the configuration files, the runtime checks only this location. If this probe fails, the runtime determines that the binding request failed and no other probing occurs.
   c. Probes for the assembly using the heuristics described in the probing section. If the assembly is not found after probing, the runtime requests the Windows Installer to provide the assembly. This acts as an install-on-demand feature.

Note There is no version checking for assemblies without strong names, nor does the runtime check in the global assembly cache for assemblies without strong names.

Figuur 17.10: Assembly Loading

Belangrijk hierin op te merken zijn volgende elementen:

- welke versie van toepassing is is afhankelijk van een aantal configuratie of policy files;
- indien geen configuratie of policy redirections gedefinieerd zijn is de at compile time gebruikte versie van toepassing, ook al is er een nieuwere versie in de GAC te vinden;
- indien de assembly al eerder werd geladen, dan wordt deze gebruikt;
- de GAC heeft voorrang op alle andere locaties. Uiteraard enkel van toepassing voor strong-named assemblies, omdat dit een voorwaarde is om in de GAC geplaatst te kunnen worden.

Eventuele uitvoeringsproblemen bij het laden van een assembly kan u documenteren met behulp van fuslogvw.exe.

17.3.4.2 Redirecting met configuration file

Een volledige write-up omtrent assembly redirection vindt u in Redirecting Assembly Versions. Onderstaande tekst is slechts een beperkte weergave van de vermeldde MSDN-entry. Het hier uitgewerkte voorbeeld maakt gebruik van de twee DLL-versies die we in de GAC plaatsten: versie 1.0.0.0 en 2.1.0.0.

- maak een WPF applicatie die refereert naar de recenste versie en gebruik daarin de klasse met het assembly versie nummer om het de assembly versie te tonen;
  - deze DLL wordt trouwens niet naar de *bin* directory gecopieerd;
- voer de toepassing uit en check de assembly versie;
- indien we de toepassing wensen te redirecten naar een andere versie (1.0.0.0) zullen we in de configuratie file wat extra XML voorzien:

The following example shows how to redirect one version of `myAssembly` to another, and turn off publisher policy for `mySecon`

```
<configuration>
   <runtime>
      <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
       <dependentAssembly>
         <assemblyIdentity name="myAssembly"
                           publicKeyToken="32ab4ba45e0a69a1"
                           culture="en-us" />
         <!-- Assembly versions can be redirected in application, publisher policy, or machine configura
         <bindingRedirect oldVersion="1.0.0.0"
                          newVersion="2.0.0.0"/>
       </dependentAssembly>
       <dependentAssembly>
         <assemblyIdentity name="mySecondAssembly"
                           publicKeyToken="32ab4ba45e0a69a1"
                           culture="en-us" />
          <!-- Publisher policy can be set only in the application configuration file. -->
          <publisherPolicy apply="no">
       </dependentAssembly>
      </assemblyBinding>
   </runtime>
</configuration>
```

Figuur 17.11: configredirection

  - het *public key token* kan u op volgende wijzes bekomen:
    * gebruik *gacutil -l*: de opgelijste informatie bevat het token;
    * zoek in de assembly cache folder (als onderdeel van diretory naam);
    * voeg een external tool toe in Visual Studio, zoals beschreven in Visual Studio Tip: Get Public Key Token for a Strong Named Assembly. Het voordeel van deze manier van werken is het copy/paste beschikbaar zijn van het token binnenin Visual Studio.

  - de *culture* tage is optioneel. Indien afwezig (of *neutral*) is de redirect van toepassing op alle cultures (taal);
  - u kan ook ranges opgeven bij het redirecten.

17.3.4.3 Redirecting met policy files

De aangeraden writeup: How to: Create a Publisher Policy Ga op volgende wijze te werk:

- wijzig de config file in de bin-directory van de WPF-applicatie zodat er geen redirection meer toegepast wordt: de gebruikte dll-versie is 2.1.0.0. Check dit door de de gecompileerde versie uit te voeren;
- wijzig de DLL-versie naar 3.0.0.0 en hercompileer de DLL (*niet* de applicatie). De toepassing werkt nog steeds met versie 2.1.0.0.
- maak een *policy.config* file in de directory waar de keypair file staat. Plaats hierin de XML-code om 2.1 te redirecten naar 3.0.
- gebruik de assembly linker *al.exe* zoals beschreven in How to: Create a Publisher Policy om een policy file aan te maken;
  - ikzelf navigeer de commandprompt met *cd* commando's naar de directory waarin zich de *snk* en *policy* file bevinden. Op deze manier vermijd ik lange filenamen (en bijhorende tikfouten) bij het intikken;
  - merkwaardig genoeg zal in de policy-file de *major.minor* van de te vervangen dll worden opgenomen, en *niet* deze van de vervangende dll die nu geïnstalleerd wordt als vervanger;
- plaats de policy file in de GAC. Om hem terug te vinden moet u in de GAC zoeken op de prefix *policy*;

```
Administrator: Visual Studio Command Prompt (2010)

D:\prevdata\PIH\vakken\20102011\programmer solutions\lessen\DemoGAC\GACDLL>al /link:policy
.config /out:policy.2.1.GACDLL.dll /keyfile:gacdll.snk /platform:anycpu
Microsoft (R) Assembly Linker version 10.0.30319.1
Copyright (C) Microsoft Corporation. All rights reserved.

D:\prevdata\PIH\vakken\20102011\programmer solutions\lessen\DemoGAC\GACDLL>gacutil /i poli
cy.2.1.GACDLL.dll
Microsoft (R) .NET Global Assembly Cache Utility.  Version 4.0.30319.1
Copyright (c) Microsoft Corporation.  All rights reserved.

Assembly successfully added to the cache

D:\prevdata\PIH\vakken\20102011\programmer solutions\lessen\DemoGAC\GACDLL>
```

Figuur 17.12: Policy file commando's

- plaats de nieuwe DLL versie in de GAC;

- test uw toepassing: deze moet nu versie 3.0 gebruiken;
- wijzig de config file terug zodat versie 1.0 gebruikt wordt. Dit illustreert de voorrang van de configuration file op de policy file.

17.3.4.4 Redirecting op machine niveau

Dit gebeurt opnieuw met configuration files en maakt geen onderdeel uit van deze cursus.

17.3.4.5 Redirecting redirections

Dit lijkt niet mogelijk te zijn:

- maak een programma dat gebruik maakt van versie 1.0 van een DLL;
- maak een policy file die versie 1.0 vervangt door versie 2.0;
- maak een configuration file die versie 2.0 vervangt door versie 3.0;
- plaats alle versies in de GAC;
- bij uitvoering wordt versie 2.0 gebruikt: herhaald redirecting lijkt dus niet te werken (wat ook overeenstemt met het artikel dat beschrijft hoe een assembly gevonden wordt)

17.3.5 GAC en project references

Voor een volledige toelichting: How to: Add or Remove References in Visual Studio. Belangrijke is het volgende element:

- GAC assemblies komen *niet* voor in de references dialogen

Dll's gevonden in de GAC worden ook niet opgenomen in de bin-directory van de client-toepassing.

17.3.6 Installeren in de GAC vanuit de Setup (tem VS 2010)

In een Visual Studio installatie project kan u een *special folder* opnemen die verwijst naar de GAC:

Figuur 17.13: GAC en setup

Indien u hier (strong named) assemblies opneemt zullen deze in de GAC van het target toestel geïnstalleerd worden.

17.4 GAC opgave

Implementeer de situatie die hierboven wordt beschreven en hieronder summier herhaald wordt:

- maak een WPF toepassing die gebruik maakt van een eigen DLL. Deze DLL biedt onder andere de mogelijkheid om zijn assembly versie te ondervragen. De applicatie gebruikt dit om de gebruikte DLL versie te tonen;
- alle versies van de DLL komen in de GAC terecht;
- gebruik een config file om te redirecten, u kiest tussen welke versies;
- gebruik een policy file om te redirecten;
- toon aan dat de configuration file redirection voorrang heeft op de policy file redirection.

17.5 Extras

addin om vanuit visual studio assembly redirection te implementeren manifest?

Jef Daels

Hoofdstuk 18

Visual Studio geautomatiseerde broncode manipulaties

18.1 Visual Studio geautomatiseerde broncode manipulaties: duiding

Een moderne IDE helpt de ontwikkelaar bij het produceren van code: zowel het reduceren van tikwerk (de IDE genereert bijvoorbeeld de proxy klasses gebruikt in een WCF-oplossing) als het vermijden van foutieve code (intellisense) zijn hiervan voorbeelden. In dit hoofdstuk komen volgende elementen aan bod: toolbox codeblocks, snippets, macro's, add-ins en (kort) het gebruik van MEF in Visual Studio 2010.

18.2 Relevante namespaces

- EnvDTE: een namespace die informatie omtrent Visual Studio projecten bevat (*alle* types projecten);
  - *Changes and new functionality are contained in the EnvDTE80, EnvDTE90, EnvDTE90a, and EnvDTE100 namespaces.*: er zijn dus upgrades
- VslangProj100: in deze assembly wordt extra informatie omtrent programmeer projecten bijgehouden die men niet kwijt kan in *EnvDTE* (bijvoorbeeld: referenties)
  - *The VslangProj100 namespace defines types used to automate project tasks. These types extend the types defined in the VSLangProj, VSLangProj2, VSLangProj80, and VslangProj90 namespaces.*: ook hier zijn dus verschillende (opeenvolgende) versie (upgrades) beschikbaar.

18.3 Toolbox codeblocks

Wanneer we in het code-window vertoeven is de toolbox meestal leeg. Het is mogelijk om een blok code in de editor te selecteren en deze op de toolbox te slepen, waarna we (via right- click) dit stukje code ook een naam kunnen geven (optioneel). Eens op de toolbox geplaatst kan u dit stukje code overal copiëren door te klikken op zijn toolbox- voorkomen:

Figuur 18.1: Code in de toolbox

18.3.1 Voordeel van toolboxcode

- zeer gemakkelijk op te zetten en te gebruiken: alle manipulaties blijven binnen Visual Studio;

18.3.2 Nadelen van toolboxcode

- de benadering is weinig flexibel: er wordt een letterijke copy aangemaakt die geen enkele ondersteuning biedt bij aanpassingen;
- deze manier van werken is niet vlot porteerbaar tussen toestellen.

Jef Daels

18.4 Code snippets

Snippets bieden de mogelijkheid om template code in een source-file te pasten. In de resultaat code is het mogelijk om intelligente vervangingen door te voeren (wijzigen op één plaats van een stukje code zal copy code, elders in het snippet ook wijzigen). Visual Studio wordt aangeleverd met tal van snippets out-of-the box en biedt de mogelijkheid om ook eigen snippets toe te voegen.

18.4.1 Snippets gebruiken

Een snippet wordt geïdentificeerd door een *shortcut* (bijvoorbeeld *propa* voor een attached dependency property). Indien u deze shortcut intikt, gevolgd door twee *TAB* toetsaanslagen wordt de gekoppelde snippet-code in uw bronfile gecopieerd:

Figuur 18.2: propa snippet gebruiken

Bemerk ook het icoontje dat Visual Studio aan een snippet koppelt. Wanneer we met deze snippet code genereren krijgen we het onderstaande resultaat:

```
public static int GetMyProperty(DependencyObject obj)
    {
    return (int)obj.GetValue(MyPropertyProperty);
    }

public static void SetMyProperty(DependencyObject obj, int value)
    {
    obj.SetValue(MyPropertyProperty, value);
    }

// Using a DependencyProperty as the backing store for MyProperty.  This enables animation, styling, bindin
public static readonly DependencyProperty MyPropertyProperty =
    DependencyProperty.RegisterAttached("MyProperty", typeof(int), typeof(ownerclass), new UIPropertyMetada
```

Figuur 18.3: propa snippet resultaat

De tekst met achtergrondkleur zal, indien hij wordt gewijzigd, ook andere tekst aanpassen

(zeker het proberen waard!). Bij het zelf aanmaken van snippets zullen we aangeven hoe deze verschillende codeelementen aan elkaar gekoppeld worden.

18.4.2 Eigen code snippets

Het *propa*-snippet genereert de dependency property registratie en de wrapper methodes. Ik zelf heb bij het definiëren van *attached dependency properties* veelal ook nood aan de propertyChanged callback, waarvan ik de signatuur niet van buiten ken: ik wou dat ik een *snippet* had die deze extra code bevatte. Het is mogelijk om de bestaande snippet aan te passen (mijns insziens minder wenselijk) of een nieuwe snippet aan te maken (een snippet is een gewone ascii-file). We zullen een eigen snippet aanmaken: *propachanged*.

U vindt de geïnstalleerde snippets in subdirectories van *Program Files\Microsoft Visual Studio 10.0\VC#\Snippets\1033*. Het *propa-* snippet wordt gevonden in de *NetFX30* subdirectory. We zullen dit snippet copiëren naar een eigen directory waarna we het snippet zullen aanpassen:

- copieer *propa.snippet* naar een eigen snippet folder;
- pas de beschikbare snippet informatie aan;
- pas de code aan en voorzie een change routine die gekoppeld wordt via de metadata, zodat de property-naam onderdeel uitmaakt van de change-routine, en mee gewijzigd wordt indien u de property naam wijzigt.

Opdat een snippet in Visual Studio gekend zou zijn moet u deze bibliotheek importeren:

- *Tools.Code Snippets Manager*;
- kies de taal waarvoor uw snippet werd ontwikkeld, en voeg uw snippet directory toe.

Hierna kan u uw zopas aangemaakt snippet testen.

18.4.3 Voordelen van snippets

- deze manier van werken kan gedeeld worden door verschillende ontwikkelaars (u vind op het internet vele voorbeelden);
- de snippet-code ondersteunt een vorm van pseudo parametristatie (en is dus iets dynamischer in gebruik dan de toolbox code);

18.4.4 Nadeel van snippets

- echte parametrisatie is nog niet mogelijk: het is bijvoorbeeld onmogelijk om een snippet te maken die in zijn textexpansie het huidige tijdstip opneemt;

Indien u de cursustekst niet leuk vond kan u ook op Extending Visual Studio Part 1 - Creating Code Snippets terecht.

18.5 Macro's in Visual Studio

Net zoals Office (Visual Basic for Applications) biedt Visual Studio de mogelijkheid om gebruikershandelingen te registereren zodat ze op een later tijdstip opnieuw uitgevoerd kunnen worden: *macro*'s. Omdat deze commando's als programma code worden bewaard kunnen ze ook aangepast worden om ook in niet identieke situaties bruikbaar te zijn. Ikzelf gebruik macro's vooral om voorbeeld code te genereren die ik dan (aangepast) overneem bij het ontwikkelen van Add-ins. Op die manier kan het onderzoeken van het Visual Studio objectmodel sterk beperkt worden (net zoals in Office worden macro's geregistreerd met Visual Basic).

18.5.1 Macro samples

Vanonder *Tools.Macros.Macro Explorer* krijgen we toegang tot de beschikbare macro's. De samples bevatten (bruikbare) voorbeeld macro's die richtinggevend zijn voor de mogelijkheden. U kan ze uitvoeren (double click) of editeren (via right click). Belangrijk hierbij is dat ze ook debugbaar zijn: u kan breakpoints plaatsen om de logica van de macro na te volgen:

- onder *DevStudio6Editor* vindt u de macro *CommentOut*. Deze zal een highlighted stuk code in commentaar plaatsen. Probeer dit in uw programma, normaal gezien zal dit niet werken. Gebruik de debugger mogelijkheden om na te pluizen waarom dit niet werkt, en pas dit aan.

18.5.2 Eigen macro's

In deze les zullen we zelf geen eigen macro's ontwikkelen maar onmiddellijk overstappen naar het ontwikkelen van addIn's.

18.5.3 Voordelen van macro's

- programmeer kennis is niet nodig om een macro op te nemen (*Tools.Macros* helpt u verder);
- een macro kan een zeer gemakkelijke manier zijn om startcode te bekomen voor het maken van een Add-In (we moeten het objectmodel van Visual Studio niet zelf uitpluizen).

18.5.4 Nadeel van macro's

- het macro recorden beperkt zich tot het registreren door middel van VB-code (voor een nMCT-student niet echt onoverkomelijk);
- macro's bieden weinig mogelijkheden tot visuele feedback. Dit is voor ons de reden om geen eigen Macro te ontwikkelen en onmiddellijk over te stappen naar Add-ins.

18.6 Visual Studio Addin's

Een Addin nestelt zich in de toolbar of het menu van uw Visual Studio omgeving en manipuleert uw Visual Studio solution. Omdat het foutief manipuleren van Visual Studio grote problemen kan veroorzaken (Visual Studio onbruikbaar kan maken) wordt in een test-fase gewerkt met een Experimental Build. Indien er iets fout loopt gebeurt dit in deze copy omgeving, en blijft uw originele Visual Studio bruikbaar.

- om addins te ontwikkelen installeert u Microsoft Visual Studio SDK;
- u kan de experimental hive (de registry entries gebruikt door de experimental build) resetten door *Visual Studio 2010 SDK.tools.Reset the Microsoft Visual Studio Experimental Hive* te kiezen.

18.6.1 Creatie van een Visual Studio Add-in

Om een Addin aan te maken gaan we als volgt te werk:

- maak een nieuw project aan met als projecttype *Other Project Types.Extensibility.Visual Studio Add-in*;
  - bij het doorlopen van de wizard kiezen we als programmeertaal *VB.NET*. Bij het programmeren van de Add-In's worden veel properties als *Object* opgeleverd. In die situaties is het handig om gebruik te maken van de late binding mogelijkheden van VB.NET;
    - u kan in *C#*een gelijkaardig effect als *late binding* bekomen door gebruik te maken van het keyword dynamic. Ikzelf vind het gemakkelijker om gewoon in VB.NET te programmeren:
      - automatische late binding;
      - de macro gegenereerde code is VBA-code, en daardoor meer lijkend op VB.NET dan op *C#*.
  - application host is de toepassing waarbinnen de add-in bruikbaar moet zijn;
  - bij het invullen van 'Choose Add-in Options' is de commentaar bij elke checkbox volgens mij duidelijk genoeg om zelfstandig de keuzes te maken;

18.6.2 Testuivoering van een Add-in

Indien u het project nu opstart wordt een nieuwe Visual Studio gestart waarin deze Addin actief is. Fouten (door de nieuwe Add-in) in deze omgeving hebben geen invloed op de originele Visual Studio omgeving.

Indien de add-in niet geladen wordt in de extra gestartte omgeving kan het eventueel helpen om in de startup options van het project volgende commandline parameter mee te geven:

- */ranu /rootsuffix Exp*

De aangemaakte Add-In bevat standaardaard al flink wat code. We zullen ons concentreren op de *Connect* file:

- de constructor van de Connect- klasse laat toe om extra initialisatie code uit te voeren;
- *OnConnection Connect!OnConnection* wordt uitgevoerd wanneer de Add-In gekoppeld wordt aan Visual Studio. Normaal gezien zullen we deze code ongewijzigd laten;
- de methode *Exec* wordt uitgevoerd wanneer de Visual Studio gebruiker de Add-In start, en bevat de eigenlijke logica van de Add-In.

18.6.3 Een lijnenteller Add-In

We zullen de add-in aanpassen zodat we in een listview per project in de solution alle files en hun aantal lijnen tonen. Met deze Add-In krijgen we een idee over de grootte van een solution.

- maak een formulier dat deze informatie zal tonen en start dit formulier vanuit de *Exec* methode (van de Connect- klasse van de Add-In);
  - momenteel is het enkel mogelijk om *WinForms* Add-in formulieren aan te maken;
  - in de ctor geeft u wellicht best het applicationobject van de addin mee (een object dat de applicatie, in dit geval Visual Studio bevat);
- *EnvDTE* is de namespace waarin u de hier nodige klasses (intefaces) terugvindt. In combinatie met de intellisense en wat zoeken kan u de lijnenteller hopelijk opzetten;
  - projectitems kunnen genest zijn: zo zijn designer generated code files projectitems van de origineel toegevoegde files;
  - niet elk projectitem heeft een overeenkomstige file;

Wees kritisch omtrent uw output: indien er maar één file gevonden wordt in uw project is er zeer veel kans dat uw Add-In foutief werkt!

18.6.4 Reference historiek Add-In

EnvDTE bevat enkel die informatie die gemeenschappelijk is aan alle Visual Studio projecten (setup, VB.NET, website, silverlight, phone, ..). Extra programmeer specifieke informatie vindt men in *VsLangProj*. Instanties van *EnvDTE* types kunnen gecast worden naar hun *VsLangProj* tegenhanger via de *.object* property van het *EnvDTE* object.

Als oefening zullen we een Add-in maken die voor elk project in een solution een file aanmaakt (datum staat in de filenaam) waarin de gebruikte referenties worden weggeschreven. Normaal gezien zou ik dit in een database registereren, maar op deze manier zijn we ook verplicht een extra file aan een project toe te voegen (een goede oefening). Met dergelijke informatie is het mogelijk te achterhalen welke projecten potentieel probleem hebben bij het verlopen of upgraden van een dll.

- maak terug een Visual Basic .NET Add-In;
- maak een macro voor de volgende acties:
  - selecteer een project in een solution;
  - voeg aan dit project een nieuwe file toe;
  - tik iets in deze nieuwe file
- in de Add-In ontwikkelen we code die per project een reference historiek file aanmaakt en daarin de references wegschrijft. Gebruik de macrocode als startcode om dit te realiseren.
  - de macro code werkt als macro code. In een Add-In werkt de code, om een nieuwe file toe te voegen niet. Gelukkig wordt in ItemOperations.AddNewItem Method duidelijk beschreven hoe de argumenten voor het toevoegen moeten geschreven worden;
  - uw code moet gebruik maken van een solution name en project name (enige nazicht in de macro code maakt duidelijk waarom). Dit komt niet automatisch, u zal hier wat code moeten voor schrijven;
- indien u de nieuw aangemaakt file wenst te saven kan u in de macro best ook een *save all* instructie opnemen.

Jef Daels

18.7 Visual Studio events

Gedurende een Visual Studio sessie doen zich vele events voor waar een Add-In kan op inpikken: het toevoegen van items, het openen en of sluiten van projecten zijn hiervan voorbeelden. Indien u hierop wenst in te pikken kan u dit op volgende wijze:

```
Private solEvents As SolutionEvents
Private tekstEvents As TextEditorEvents
Private Sub startupComplete()
    solEvents = _applicationObject.Events.SolutionEvents
    AddHandler solEvents.ProjectAdded, AddressOf projAdded
    AddHandler solEvents.Opened, AddressOf solOpened

    tekstEvents = _applicationObject.Events.TextEditorEvents
    AddHandler tekstEvents.LineChanged, AddressOf LineChanged
End Sub
Private Sub solOpened()
    MessageBox.Show("sol opened")
End Sub
Private Sub projAdded(ByVal proj As EnvDTE.Project)
    MessageBox.Show("proj added")

End Sub
Private Sub LineChanged(ByVal start As EnvDTE.TextPoint, ByVal eind As Env
    MessageBox.Show("item added")
End Sub
```

Figuur 18.4: Eventhandling in een Add-In

Belangrijk in bovenstaande code zijn volgende elementen:

- de events- variabelenw waarvoor events worden opgevangen moeten op moduleniveau geplaatst worden (zie ook Getting DocumentEvents after solution is opened).
- het koppelen van events gebeurt in VB.NET niet door middel van de *+=*- operator maar met behulp van een *AddHandler*- instructie. Uiteraard moet ook hier de signatuur van de gekoppelde routine geschikt zijn;
- het grootste probleem is wellicht het vinden van het gepaste event block. In het voorbeeld komen volgende zaken aan bod:
  - het openen van een solution;
  - het toevoegen van een project
  - het wijzigen van een lijn code

18.8 Deployment van een Add-In

Indien een Add-In naar behoren werkt wensen we deze niet langer in de experimental hive uit te voeren: we wensen de Add-In te gebruiken in een normale Visual Studio omgeving, of de add-In te verspreiden naar andere computers. Om dit te realiseren gaan we als volgt te werk:

- lees How to create and load add-ins in Visual Studio .NET by using Visual $C^{\#}$.NET or in Visual Studio 2005 by using Visual $C^{\#}$2005 en voer uit wat daarin beschreven staat (er ontbreken belangrijke stappen, maar een goed begrijper met voorkennis kan die wel oplossen)
  - pas deze manier van werken toe voor de references Add-In;
  - bij mij werkt het niet;
- de ouderwetse manier (zonder .msi file) die enkel werkt indien u Visual Studio installeerde (omdat we de referenties niet meeinstalleren):
  - via *Tools.Options* achterhalen we de directories waarin Add-Ins gezocht worden:

Figuur 18.5: Addin directories

  - in mijn installatie bestond de *Addins*- directory niet: ik heb deze manueel aangemaakt

Jef Daels

  - copieer de *.addin* file die Visual Studio in uw bronfile (bij de code files) plaatste, tesamen met de *.dll* uit de compilatie directory naar de Add-In directory die u koos (indien u niet één van de aanwezige directories koos moet u uw eigen directory toevoegen)
  - bij het heropstarten zal visual studio de AddIn terugvinden en integereren;
- u kan het al dan niet activeren van Add-Ins aansturen vanonder *Tools.Add-in Manager*.

18.9 Visual Studio en MEF: extentions

Zoals in het hoofdstuk MEF werd aangehaald is Visual Studio 2010 MEF-gebaseerd. De technologie op zich is relatief gemakkelijk te begrijpen, maar het voorzien van extenties in een complexe software als Visual Studio is toch niet evident. Extending the Editor is een potentieel startpunt om u hierin te verdiepen. In het vervolg van deze tekst zullen we de intellisense uitbreiden om een gemakkelijk commentaarregels aan code toe te voegen. Uiteindelijk wordt ook toegelicht hoe deze extenties kunnen *gedeployed* worden.

Hoewel we nu enkel zullen ingrijpen op de intellisense kan er natuurlijk veel meer gerealiseerd worden. Een aantal voorbeelden (met broncode) hiervan kan u terugvinden op MSDN Developer Samples: Learn Through Code waarbij u met behulp van het zoekvenster zoekt op *VSX*

18.9.1 cmnt extention

Visual Studio gebruikt intellisense om de ontwikkelaar bij het coderen mogelijke aanvullingen te presenteren. Dit verhoogt de tiksnelheid, vermijdt fouten en vereist gememoriseerde namen van de ontwikkelaar. Het is de bedoeling om via intellisense een commentaarregel te kunnen toevoegen waarin zowel de naam van de ontwikkelaar als het tijdstip van toevoegen voorkomt:

Figuur 18.6: Cmt intellisense

- dit kan niet met snippets gerealiseerd worden omdat deze uit vaste tekst bestaan: de auteur en het tijdstip zouden vaste waardes zijn die achteraf gewijzigd moeten worden.

18.9.2 cmt extention implementatie

- Walkthrough: Displaying Statement Completion diende als leidraad om mijn eigen *cmnt* completion aan te maken: volg de volledige beschrijving;

- het voorbeeld past intellisense toe op *PlainText* files en werkt dus enkel op txt-files, en niet in $C^{\#}$of VB- code files. Om dit te verhelpen moet u het *ContentType* aanpassen (op twee! plaatsen);
  - natuurlijk moet u bij het overnemen van de code de gepaste wijzigingen aanbrengen: u moet de *completions* van het voorbeeld aanpassen. Wie de code begrijpt zal daar weinig problemen bij ondervinden;
  - u zal moeten opzoeken wel ContentType u dan wel dient te gebruiken.
- ik heb niet de pretentie die code voldoende te begrijpen om ze zelf van scratch te ontwikkelen.

18.9.3 cmt deployment

Indien u een extention wenst te installeren is copy deployment onvoldoende. Er is nood aan een *VSIX*- file, die bij dubbelklikken de extentie installeert. Het aanmaken van een VSIX-file wordt beschreven in Packaging an Extension in an Empty VSIX Project. Beter dan die telst kan ik de topic niet beschrijven.

18.9.4 Extentions en Visual Studio

Visual Studio heeft een *Tools.Extention Manager*. Hiermee kunnen geïnstalleerde extentions gedisabled of uninstalled worden.

18.10 Relevante literatuur

- Creating and Using IntelliSense Code Snippets: alles wat u ooit over snippets wou weten, en nog veel meer!
- Getting Started with Extending Visual Studio
- Extending the Editor
- VSX Home on Code Gallery

18.11 Codemanipulatie opgaves

- ontwikkel een snippet waarmee een attached dependency property wordt aangemaakt, aangevuld met een change- handler. De change handler routine naam is gekoppeld aan de property naam (en wordt dus gewijzigd indien de property naam wordt gewijzigd);
- ontwikkel een Add-In om voor elke file in elk project van een solution het aantal codelijnen te tellen;
- ontwikkel een Add-In om voor elk project in een solution de referenties in een file bij te houden. Elke uitvoering maakt een eigen versie van deze file. U neemt de datum en het tijdstiop hiervoor op in de filenaam;
- (optioneel) maak een extention die toelaat om een commentaar lijn (zoals hierboven beschreven) toe te voegen aan een code window.

Jef Daels

Figuren

2.1 Extention method definition . . . 13
2.2 Extention method gebruik . . . 13
2.3 Extention method gebruik: fadeto . . . 14
2.4 Await demo . . . 15
2.5 Await demo async methode . . . 15

3.1 De klasse nMCTDocent . . . 17
3.2 DisplayMemberPath binding . . . 18
3.3 BindingPathError . . . 18
3.4 ILSpy DisplayMemberPath info . . . 19
3.5 ViewModel klasse . . . 20
3.6 Application Resources . . . 20
3.7 Window binding . . . 20
3.8 DataContext binding . . . 21
3.9 ItemsSource declaratie . . . 21
3.10 Observable hoofding . . . 22
3.11 Collection hoofding . . . 22
3.12 ItemsSource binding . . . 22
3.13 ItemTemplate instellen . . . 22
3.14 nMCTDocent datatemplate . . . 23
3.15 nMCTDocenten in een ComboBox . . . 23
3.16 RelativeSource binding . . . 24

4.1 Delegate definitie . . . 26
4.2 Delegate demo: uppercase en lowercase . . . 26
4.3 Event definitie (zonder guidelines) . . . 28
4.4 Event abonnering . . . 28
4.5 Event afvuren (zonder guidelines) . . . 29
4.6 Static event definitie (volgens de guidelines) . . . 31
4.7 Static event client . . . 31
4.8 Asynchrone verwerking demo . . . 34
4.9 AsyncCallback demo . . . 35
4.10 UI-thread problem . . . 35
4.11 Dispatcher demo . . . 36
4.12 Await- keyword . . . 37

4.13 XMLViewer en code plumbing . . . 40
4.14 Anonymous method voorbeeld . . . 42
4.15 Lambda expressie voorbeeld . . . 42

5.1 ComponentModel attributen ($C^{\#}$) . . . 46
5.2 Assembly reflection ($C^{\#}$) . . . 47
5.3 Strongly typed instantiatie . . . 48
5.4 Activator.CreateInstance ($C^{\#}$) . . . 48
5.5 Activator.InvokeMember ($C^{\#}$) . . . 49

6.1 Commandline compilatie: $C^{\#}$broncode . . . 54
6.2 $C^{\#}$commandline compilatie en uitvoering . . . 55
6.3 Commandline compilatie: VB.NET broncode . . . 55
6.4 VB.NET commandline compilatie voorbeeld . . . 56
6.5 $C^{\#}$System.CodeDom.Compiler namespace . . . 57
6.6 CodeDOM . . . 58
6.7 ILDasm output . . . 59
6.8 ILSpy output . . . 60
6.9 Reflector output . . . 61
6.10 Obfuscated output via Reflector . . . 62
6.11 Tekenen van een functie . . . 63

7.1 NQueens oplossen . . . 66
7.2 Sudoku spelregels: een opgave . . . 67
7.3 Sudoku spelregels: een oplossing . . . 68
7.4 Sudoku routed events demo . . . 70
7.5 Awaiting een oproep . . . 71
7.6 TaskCompletionSource: SetResult . . . 71
7.7 TaskCompletionSource demo . . . 72
7.8 4 op een rij: bordwaarde . . . 73
7.9 Minimax algoritme . . . 75

8.1 Attached dependency property in XAML . . . 80
8.2 Attached dependency property definitie ($C^{\#}$) . . . 81
8.3 F1 infrastructuur ($C^{\#}$) . . . 82
8.4 CLR binding error . . . 84
8.5 Dependency property syntax 1 ($C^{\#}$) . . . 84
8.6 Dependency property syntax 2 ($C^{\#}$) . . . 85
8.7 Breedte animatie ($C^{\#}$) . . . 85
8.8 Background animatie . . . 86
8.9 Transformatie van een Button ($C^{\#}$) . . . 87
8.10 Transformatie animatie ($C^{\#}$) . . . 88
8.11 ValueConverter definitie ($C^{\#}$) . . . 89
8.12 ValueConverter XAML ($C^{\#}$) . . . 90
8.13 XAML binding dependency property . . . 92

9.1 Window met eigen caption . . . 94
9.2 Window met WindowStyle none . . . 95
9.3 Checkbox states visualisatie . . . 98
9.4 Checkbox states via Blend . . . 98
9.5 State animaties in XAML . . . 99
9.6 State en Parte opgave: verkeerslicht . . . 99

10.1 Height en applicationsettings . . . 105
10.2 Enumeration definitie . . . 107

11.1 Sudoku in PDF . . . 110

12.1 Kwadraat code . . . 114
12.2 Kwadraten selecteren . . . 115

14.1 AOP: parameter annotatie . . . 123
14.2 PostSharp en ILSpy . . . 125
14.3 PDFStamper performance counter . . . 127
14.4 Performance counters in Visual Studio . . . 128

17.1 Global Assembly Cache (iets oudere versie, Win 7) . . . 136
17.2 Global Assembly Cache (iets oudere versie, Win 8) . . . 136
17.3 Eigen GAC-dll's, Win 8) . . . 137
17.4 Strong-named assemblies (MSDN) . . . 138
17.5 Gacutil -i Strong-named failure . . . 139
17.6 Gacutil -i Strong-named success . . . 139
17.7 Gacutil -l . . . 139
17.8 Assembly versies . . . 140
17.9 File properties . . . 140
17.10Assembly Loading . . . 141
17.11configredirection . . . 142
17.12Policy file commando's . . . 143
17.13GAC en setup . . . 145

18.1 Code in de toolbox . . . 148
18.2 propa snippet gebruiken . . . 149
18.3 propa snippet resultaat . . . 149
18.4 Eventhandling in een Add-In . . . 155
18.5 Addin directories . . . 156
18.6 Cmt intellisense . . . 157

Index

Action, 41
Activator, 48
  CreateInstance, 48
Add-ins, 152
AddHandler, 96, 155
addOwner, 83
AddressOf, 29
AffectsMeasure, 91
AffectsRender, 91
al.exe, 143
Alfa Beta pruning, 75
Animaties, 85
Animation
  ColorAnimation, 86
  DoubleAnimation, 85
  Transformation, 87
Anonymous method, 42
AOP, 123
  ContextBoundObject, 123
Application settings, 105
Assembly, 45
Async, 37
async, 15
AsyncCallback, 34
Asynchrone verwerking, 34
  UI-thread, 36
Attach to proces, 116
Attached dependency property, 79
Attribute
  Category, 46
  DefaultEvent, 46
  DefaultProperty, 46
  DefaultValue, 46
  Description, 46
Await, 37
await, 15
backtracking, 65
BeginInvoke, 34
Behavior, 103
  Drag, 102
  Grip, 103, 105
Behaviors, 101
BesteZet, 74
Blend, 88
Bootstrapper Project, 121
BordWaarde, 73
Brush, 86
bubbling, 32

CanConvertFrom, 90
Category, 46
ccCaption, 95
Closed, 106
cmnt token, 157
Code snippet, 149
CodeDomProvider, 57
Color, 86
ColorAnimation, 86
CompilerParameters, 57
CompilerResults, 58
Connect, 153
  Connect, 153
  Exec, 153
ContextBoundObject, 123
contravariance, 27
Convert, 89
ConvertBack, 89
ConvertFrom, 90
covariance, 27
CreateInstance, 48
CreateProvider, 57

DataContext, 19
DefaultEvent, 46

DefaultProperty, 46
DefaultValue, 46
Delegate, 25, 45
  Asynchrone verwerking, 34
  BeginInvoke, 34
  codeplumbing, 39
  events, 27
  multithreading, 33
Dependency properties, 79
DependencyProperty, 81
  addOwner, 83
  RegisterAttached, 81
Description, 46
deserialisatie, 89
Dispatcher, 36
DisplayMember, 18
DisplayMemberPath, 18
DoubleAnimation, 85

EAP, 37
EnvDTE, 147, 153
Event based Asynchronous Pattern, 37
EventArgs, 30
Events
  +=, 28
  AddHandler, 29
  AddressOf, 29
  event keyword, 28
  EventArgs, 30
  guidelines, 30
  handles, 28
  onEvent, 30
  preview, 32
  RaiseEvent, 29
  routed events, 32
  static events, 31
  tunneling event, 32
Events:bubbling event, 32

F1URL, 80
Func, 41

GAC, 135
Gacutil.exe, 139
Generics, 30
GetCustomAttributes, 49
GetLanguageFromExtension, 57
GetTemplateChild, 96
Ghost data, 33
Global Assembly Cache, 135
Grip, 103

IAOPValidatie, 125
IAsyncResult, 34
IL, 58, 59
ILDasm, 59
inconsistent read, 33
InstallShield, 119
instructies
  lock, 33
intermediate language, 59
InvokeMember, 49
ItemsControl, 21, 22
ItemsSource, 21
ItemsTemplate, 22
iTextSharp, 109
IValueConverter, 89, 90
  Convert, 89
  ConvertBack, 89

Lambda expression, 42
LinearGradientBrush, 89
LINQ, 42
Lost update, 33

Macros, 151
Main, 54
MEF, 157
MenuItem, 52
MiniMax, 73
  BesteZet, 74
  BordWaarde, 73
  MiniMax, 75
  Pruning, 75
MouseDownEvent, 102
MouseDragElementBehavior, 102
MouseMoveEvent, 102
MSI, 119
Multithreading, 33

N-Queen, 66

Namespace
  System, 25
  System.CodeDom.Compiler, 53
  System.ComponentModel, 45, 46, 79
  System.Reflection, 45
  System.Windows, 79
  System.Windows.Data, 79
  System.Windows.Media, 79
nMCTDocent, 17

Obfuscation, 62
ObservableCollection, 17
OnApplyTemplate, 94, 96
OnEntry, 124, 125
OnException, 125
OnMethodBoundaryAspect, 124
OnRender, 91
OpenFileDialog, 47

PARTS, 94
patterns
  strategy pattern, 39
Performance counter, 127
PerformanceCounterType, 128
plugin, 45
policy, 137
PostSharp, 123, 124
  OnEntry, 124
  OnException, 125
  OnMethodBoundaryAspect, 124
preview, 32
PreviewKeyDownEvent, 69
Process, 82
propa, 149
propachanged, 150
PropertyPath, 86
Pruning, 75

Query Notification, 131

redirect, 137
Reflection, 45
Reflector, 61
registry, 135
Resize, 104
runtime compilatie, 53
RuntimeInitialize, 126

Server Explorer, 128
signatuur, 27
sn.exe, 138
Snippet, 149
  propa, 149
snippet
  propa, 81
  propdp, 81
snk, 138
SQL Server, 113
SQL Server Datatools, 113
States, 97
StoryBoard, 86
  TargetNameProperty, 86
  TargetPropertyProperty, 86
String, 54
Strong name, 138
Sudoku, 110
System, 25, 54
System.CodeDom.Compiler, 53, 57
System.ComponentModel, 45, 46, 79
System.Reflection, 45
System.Windows, 79
System.Windows.Data, 79
System.Windows.Media, 79

Table Value Functions, 115
Task, 71
TemplatePart, 94
Themes, 96
Thread
  Sleep, 38
Toolbox code, 148
Transformation, 87
TryParse, 90
tunneling, 32
Type.GetCustomAttributes, 49
Type.InvokeMember, 49
TypeConverter, 89, 90

UI-thread, 36
using, 54

ValueConversion, 89

Jef D

Version, 139
Visual Studio, 128
VisualStateManager, 99
VisualTreeHelper, 96, 103
  GetParent, 96
VSIX, 158
VslangProj100, 147

Window, 50
WindowStyle, 95
WiX, 119

XMLViewer, 39

Programmer solutions

Jef Daels

at werkt!!!

Jef Daels
lol, dat werkt!!!

Inhoud

1 Cursus aanpassingen **11**

2 Random topics **13**
2.1 Extention methods . . . 13
2.1.1 Gebruik van Extention methods . . . 14
2.1.1.1 LINQ en extention methods . . . 14
2.1.1.2 Animaties en extention methods . . . 14
2.1.2 Extention method surprises . . . 14
2.2 Partial classes . . . 14
2.3 Partial methods . . . 14
2.4 Transformations . . . 14
2.5 Awaitable methodes . . . 15
2.6 (XML) serialisatie in .NET . . . 15
2.7 Var keyword . . . 16
2.8 TryParse methodes . . . 16
2.9 Dynamic keyword . . . 16
2.9.1 lock en volatile keywords . . . 16
2.9.2 WPF toolkit . . . 16
2.9.3 Application settings in WPF . . . 16
2.9.4 conditional[debug] . . . 16
2.9.5 custom build . . . 16
2.9.6 sys tools . . . 16

3 Op aanvraag: Data Binding in WPF **17**
3.1 De klasse nMCTDocent . . . 17
3.2 DisplayMemberPath: the old way . . . 18
3.3 DataContext: default binding source . . . 19
3.3.1 DataContext: duiding . . . 19
3.3.2 Datacontext: instellen op een window . . . 20
3.3.2.1 ViewModel voor de demo . . . 20
3.3.2.2 Instellen van de Window DataContext . . . 20
3.3.2.3 Binden van de ItemsControl.ItemsSource Property . . . 21
3.3.2.4 Instellen van de ItemsControl.ItemTemplate Property . . . 22
3.4 Niet DataContext binding sources . . . 23
3.5 TypeConverter . . . 24

lol, dat werkt!!!

4 Delegates **25**
4.1 Delegates: duiding . . . 25
4.1.1 Relevante namespace . . . 25
4.2 Delegates: technische intro . . . 26
4.2.1 Demo-code en toelichting . . . 26
4.2.2 Covariance en contravariance . . . 27
4.2.2.1 Covariance . . . 27
4.2.2.2 Contravariance . . . 27
4.3 Delegates en events . . . 27
4.3.1 Definitie van een event . . . 28
4.3.2 Abonneren (to subscribe) op een event . . . 28
4.3.2.1 Event subscription: verschil met VB.NET . . . 29
4.3.3 Afvuren van een event . . . 29
4.3.3.1 Afvuren van een event: verschil met VB.NET . . . 29
4.3.4 Meerdere clients voor hetzelfde event . . . 30
4.4 Event Guidelines! . . . 30
4.5 Omtrent Events: . . . 31
4.5.1 Static events . . . 31
4.5.2 Delegates of Interfaces: .NET of JAVA events . . . 32
4.5.3 Routed events . . . 32
4.6 Delegates en multithreading . . . 33
4.6.1 Multithreading: wat bedoelt u? . . . 33
4.6.2 Multithreading: iets voor ons? . . . 33
4.6.3 Asynchrone processing en delegates . . . 34
4.6.3.1 Asynchroon, geen AsyncCallback . . . 34
4.6.3.2 Asynchroon, met AsyncCallback, zonder UI . . . 35
4.6.3.3 Asynchroon, met AsyncCallback, met UI . . . 36
4.6.4 Multithreading met de Thread klasse . . . 36
4.6.5 Event based Ansynchronous Pattern (EAP) . . . 37
4.6.6 Asynchroon programmeren made easy . . . 37
4.6.7 CancellationToken . . . 38
4.6.8 Thread.Sleep . . . 38
4.6.9 Meer info? . . . 38
4.7 Code plumbing, aka 'the strategy pattern' . . . 39
4.7.1 Codeplumbing: een XMLViewer . . . 39
4.7.2 Codeplumbing pitfalls . . . 40
4.8 Delegates: faits divers . . . 41
4.8.1 Action en Func . . . 41
4.8.1.1 Action . . . 41
4.8.1.2 Func . . . 41
4.8.2 Meer dan (begin)-invoke . . . 41
4.8.3 Timers . . . 41
4.8.4 Asynchrone eventverwerking . . . 42
4.8.5 Anonymous methods (.NET 2.0) . . . 42

Jet Delegates
lol, dat werkt!!!

4.8.6 Lambda expressions (.NET 3.0) . . . 42
4.9 Delegates: niet voor elk programma een geschikte oplossing . . . 43
4.10 Delegates: opgaves . . . 43

5 Reflection **45**
5.1 Reflection: duiding . . . 45
5.1.1 Relevante namespaces . . . 45
5.2 Reflection: attributen voorbeeld . . . 46
5.3 Reflection in .NET: meer dan attributen . . . 47
5.3.1 Reflection in .NET: start . . . 47
5.3.2 Hoe een instantie maken van een klasse in een assembly? . . . 48
5.3.3 .NET en attributes . . . 49
5.3.3.1 .NET en attribute informatie opvragen . . . 49
5.3.3.2 Hoe zelf Attribute klasses maken in .NET? . . . 49
5.4 Een eigen plugin- infrastructuur! . . . 50
5.4.1 Een eigen plugin- infrastructuur: vereisten . . . 50
5.4.2 Een eigen plugin- infrastructuur: mogelijke oplossing . . . 50
5.4.3 Een eigen plugin- infrastructuur: implementatie . . . 51
5.4.4 Eigen plugin infrastructuur: demo . . . 52
5.5 Namespaces met interessante attributen . . . 52
5.6 Reflection: opgaves . . . 52

6 Runtime compilatie **53**
6.1 Runtime compilatie: duiding . . . 53
6.1.1 Relevante namespaces . . . 53
6.2 .NET compilaties starten . . . 54
6.2.1 Command line compilatie . . . 54
6.2.1.1 Command line compilatie: $C^{\#}$ . . . 54
6.2.1.2 Command line compilatie: VB.NET . . . 55
6.2.1.3 Een extraatje omtrent MSBuild . . . 56
6.2.2 Runtime compilatie: System.CodeDom.Compiler . . . 57
6.3 CodeDom . . . 58
6.4 System.Reflection.emit . . . 58
6.5 Het Roslyn project . . . 58
6.6 Assembly code inspecteren . . . 59
6.6.1 ILDasm . . . 59
6.6.2 ILSpy . . . 60
6.6.3 Reflector . . . 61
6.6.4 Obfuscatie . . . 62
6.6.4.1 Confuser . . . 62
6.7 Opgaves . . . 63
6.7.1 Een tekenprogramma . . . 63
6.7.2 ILSpy . . . 63
6.7.3 Obfuscatie . . . 63

lol, dat werkt!!!

7 Backtracking **65**
7.1 Backtracking: duiding . . . 65
7.1.1 Relevante namespaces . . . 65
7.1.2 Backtracking: principe . . . 65
7.2 N-Queens probleem . . . 66
7.2.1 Probleemstelling, synchrone oplossing . . . 66
7.2.2 Asynchroon: BeginInvoke/4 . . . 66
7.2.3 Asynchroon: async/await . . . 67
7.3 Sudoku's oplossen . . . 67
7.3.1 Sudoku: spelregels . . . 67
7.3.2 Software vereisten . . . 68
7.3.3 Sudoku oplos algoritme . . . 68
7.3.3.1 Constraint satisfaction solver . . . 69
7.4 Sudoku routed events introductie . . . 69
7.5 Async methodes: TaskCompletionSource . . . 70
7.5.1 Task- klasses . . . 71
7.5.2 TaskCompletionSource- klasses . . . 71
7.6 Het MiniMax algoritme (geen examen stof) . . . 73
7.6.1 MiniMax benadering voor vier op een rij . . . 73
7.6.1.1 BordWaarde . . . 73
7.6.1.2 BesteZet . . . 74
7.6.1.3 MiniMax methode . . . 75
7.6.2 Minimax optimalisaties . . . 75
7.6.2.1 Alfa-Beta pruning . . . 75
7.6.2.2 Multi- threading . . . 76
7.7 Backtracking opgaves . . . 77

8 Dependency properties **79**
8.1 Dependency properties: duiding . . . 79
8.1.1 Relevante namespaces . . . 79
8.2 F1URL: een help infrastructuur . . . 80
8.2.1 F1URL gebruik in XAML . . . 80
8.2.2 F1URL definitie . . . 81
8.2.3 F1URL infrastructuur . . . 82
8.3 Dependency properties: technisch . . . 83
8.4 Dependency property XAML binding . . . 84
8.5 Dependency property syntax . . . 84
8.6 Dependency properties en animaties . . . 85
8.6.1 DoubleAnimation . . . 85
8.6.2 ColorAnimation . . . 86
8.6.3 Transformatie animaties . . . 87
8.6.4 Animaties en XAML . . . 88
8.6.5 $C^{\#}$of XAML animaties? . . . 88
8.7 DependencyProperties in XAML: TypeConverters . . . 89

lol, dat werkt!!!

8.7.1 IValueConverter interface . . . 89
8.7.2 Default dataconverters . . . 90
8.7.2.1 TryParse . . . 90
8.8 Dependency properties en OnRender . . . 91
8.9 Veel voorkomende fouten . . . 91
8.10 Extra lectuur . . . 91
8.11 Dependency property: opgaves . . . 92

9 Templated controls **93**
9.1 Templated controls: duiding . . . 93
9.2 Relevante namespaces . . . 93
9.3 CaptionControl . . . 94
9.3.1 PARTS . . . 94
9.3.1.1 Caption control via PARTS: inleiding . . . 94
9.3.1.2 Caption Control via PARTS: uitwerking . . . 95
9.3.1.3 Caption control via Parts: test en aanpassing . . . 96
9.3.1.4 Caption Control via PARTS: evaluatie . . . 97
9.4 Caption Control opgave . . . 97
9.5 States . . . 97
9.6 State opgave . . . 99

10 Behaviors **101**
10.1 Behaviors: duiding . . . 101
10.2 Relevante namespaces . . . 101
10.3 Drag behavior . . . 102
10.3.1 Drag behavior: dependency property versie . . . 102
10.3.2 Drag behavior: Blend out of the box versie . . . 102
10.3.3 Drag behavior: gespecialiseerde Blend versie . . . 103
10.4 Grip behavior . . . 103
10.5 Resize behavior . . . 104
10.5.1 Alignment impact (Laurenz Demey 2010- 2011) . . . 104
10.6 Behavior opgaves . . . 105
10.7 WPF en application settings . . . 105
10.8 Enumerations: getalwaardes en bewerkingen . . . 107
10.9 Extra lectuur . . . 108

11 iTextSharp **109**
11.1 Installatie . . . 109
11.2 PDF creatie (en opgaves) . . . 109
11.3 Manipulatie van een bestaande PDF (en opgaves) . . . 110
11.4 Tekenen (en opgaves) . . . 110
11.5 Toepassingsgebied voor iText . . . 111

lol, dat werkt!!

12 SQLServer en .NET integratie **113**
12.1 Nodige softwares . . . 113
12.2 Kwadraten berekenen . . . 114
12.3 SQL Server tabellen raadplegen . . . 115
12.4 Debugging SQLServer . . . 116
12.4.1 Debugging startend vanuit het Visual Studio project . . . 116
12.4.2 Debugging van een SQL Server call gestart buiten het project . . . 116
12.5 Theorie vragen . . . 117

13 Windows installer oplossingen **119**
13.1 Demo programma . . . 119
13.2 InstallShield Limited Edition . . . 120
13.2.1 Download en installatie . . . 120
13.2.2 Aanmaken van een MSI-pakket . . . 120
13.2.3 Redistributables . . . 120
13.2.4 Evaluatie InstallShield Limited Edition . . . 120
13.3 WiX . . . 121
13.3.1 Download en installatie . . . 121
13.3.2 Aanmaken van een MSI-pakket . . . 121
13.3.3 Redistributables . . . 121
13.3.4 Evaluatie WiX . . . 121
13.4 Welke installer? . . . 122
13.4.1 Vergelijking InstallShield en Wix . . . 122
13.4.2 Vergeet niet: . . . 122
13.5 Voorbeeld labo examen opgave . . . 122

14 AOP: Aspect Oriented programming **123**
14.1 AOP duiding . . . 123
14.1.1 Relevante namespaces . . . 123
14.2 PostSharp . . . 124
14.2.1 PostSharp kennismaking: methode-entry teller . . . 124
14.2.2 PostSharp en parameter annotaties . . . 125
14.2.3 PostSharp en performance counters . . . 127
14.2.3.1 Eigen performance counters . . . 128
14.2.3.2 AOP en performance counters . . . 128
14.3 AOP oefeningen . . . 129

15 Query Notification **131**
15.1 Query notification oefening . . . 131

16 Discussie topics **133**
16.1 Duiding . . . 133
16.2 Sinus berekenen mbv Taylor reeks . . . 134

lol, dat werkt!!!

17 Global Assembly Cache: GAC **135**
17.1 Global Assembly Cache: duiding . . . 135
17.1.1 Relevante namespaces . . . 135
17.2 GAC: technische duiding . . . 136
17.3 Ontwikkelen voor en met de GAC . . . 138
17.3.1 Strong named assemblies . . . 138
17.3.2 Gacutil.exe . . . 139
17.3.3 DLL-versies maken . . . 140
17.3.4 Redirecting assemblies . . . 141
17.3.4.1 Welke assembly wordt geladen? . . . 141
17.3.4.2 Redirecting met configuration file . . . 142
17.3.4.3 Redirecting met policy files . . . 143
17.3.4.4 Redirecting op machine niveau . . . 144
17.3.4.5 Redirecting redirections . . . 144
17.3.5 GAC en project references . . . 144
17.3.6 Installeren in de GAC vanuit de Setup (tem VS 2010) . . . 144
17.4 GAC opgave . . . 145
17.5 Extras . . . 146

18 Visual Studio geautomatiseerde broncode manipulaties **147**
18.1 Visual Studio geautomatiseerde broncode manipulaties: duiding . . . 147
18.2 Relevante namespaces . . . 147
18.3 Toolbox codeblocks . . . 148
18.3.1 Voordeel van toolboxcode . . . 148
18.3.2 Nadelen van toolboxcode . . . 148
18.4 Code snippets . . . 149
18.4.1 Snippets gebruiken . . . 149
18.4.2 Eigen code snippets . . . 150
18.4.3 Voordelen van snippets . . . 150
18.4.4 Nadeel van snippets . . . 150
18.5 Macro's in Visual Studio . . . 151
18.5.1 Macro samples . . . 151
18.5.2 Eigen macro's . . . 151
18.5.3 Voordelen van macro's . . . 151
18.5.4 Nadeel van macro's . . . 151
18.6 Visual Studio Addin's . . . 152
18.6.1 Creatie van een Visual Studio Add-in . . . 152
18.6.2 Testuivoering van een Add-in . . . 152
18.6.3 Een lijnenteller Add-In . . . 153
18.6.4 Reference historiek Add-In . . . 153
18.7 Visual Studio events . . . 155
18.8 Deployment van een Add-In . . . 156
18.9 Visual Studio en MEF: extentions . . . 157
18.9.1 cmnt extention . . . 157

lol, dat werkt!!!

18.9.2 cmt extention implementatie 157
18.9.3 cmt deployment 158
18.9.4 Extentions en Visual Studio 158
18.10 Relevante literatuur 158
18.11 Codemanipulatie opgaves 159

lol, dat werkt!!!
Jeroen Aels

Hoofdstuk 1

Cursus aanpassingen

- templated controls: een no-code kalender (2 views: 3 cbo's, en een maandgrid met navigatie)
- content uit sexy $C^{\#}$

lol, dat werkt!!!

Jef Daels
lol, dat werkt!!!

Hoofdstuk 2

Random topics

2.1 Extention methods

Extention methods bieden de mogelijkheid om, op syntactische wijze, methodes toe te voegen aan klassen: u kan met behulp van de .- notatie deze methodes oproepen. Dit is enkel *syntactic sugar*, omdat de klasse definitie zelf niet wordt uitgebreid (hiervoor zou u overerving nodig hebben).

Het definiëren van een extention method gebeurd door middel van een static routine, zoals hieronder met een voorbeeld beschreven:

werkt!!

```
public static int AantalKlinkers(this string tekst)
    {
    int iAantal = 0;
    foreach (char c in tekst)
        if (IsKlinker(c)) iAantal++;
    return iAantal;
    }
```

Figuur 2.1: Extention method definition

Bij het coderen van een extention method zal de intellisense tonen dat het een extention method betreft:

```
public MainWindow()
    {
    InitializeComponent();
    Console.WriteLine("aantal klinkers in " + this.Title + ": "
        + this.Title.AantalKlinkers());
    int i = this.Title.aantal
    }
}
```

AantalKlinkers | (extension) int string.AantalKlinkers()
All | Tokens

Figuur 2.2: Extention method gebruik

2.1.1 Gebruik van Extention methods

2.1.1.1 LINQ en extention methods

Extention methods werden geïntroduceerd tesamen met LINQ, wiens implementatie grotendeels gebaseerd is op Extention methods (zie de intellisense bij het oproepen van een LINQ-methode)

2.1.1.2 Animaties en extention methods

Designers zullen animaties beschrijven door middel van XAML code (bijvoorbeeld bij een STATE-wijziging bij het gebruik van de VisualStateManager). Eenzelfde animatie zal op verschillende plaatsen gedupliceerd worden. Indien we de animatie als extention method implementeren kan ze eenvoudig (via code) herhaaldelijk worden toegepast:

```
spNumbers11.FadeTo((speler1Waits) ? 0.15 : 1.0, 500);
spNumbers12.FadeTo((speler1Waits) ? 0.15 : 1.0, 500);
spNumbers21.FadeTo((!speler1Waits) ? 0.15 : 1.0, 500);
spNumbers22.FadeTo((!speler1Waits) ? 0.15 : 1.0, 500);
```

Figuur 2.3: Extention method gebruik: fadeto

Opmerking: de gecodeerde animatie is gemakkelijk in een behavior te plaatsen zodat ze ook vanuit XAML snel kan worden geactiveerd.

Jeff Dwerks lol, dat werkt

2.1.2 Extention method surprises

Extention methodes gedefinieerd op een klasse zijn ook van toepassing op instanties van een ervende klasse.

2.2 Partial classes

2.3 Partial methods

2.4 Transformations

Een transformatie kan opgebouwd zijn uit verschillende elementen: een draaiing (rotatie) een schaling (scaling), een verplaatsing (translation) en kanteling (skewing). Indien u een tranformatie op een object toepast dient u rekening te houden met de reeds bestaande transformaties: indien u een object wenst te herschalen mag u de eventuele draaiing niet verwijderen. Een CompositeTransform object zal in veel gevallen uitermate geschikt zijn om met één object al uw tranformaties te defini¨eren en animeren. Dit vermijdt dat u voor een animatie alle TransformGroup onderdelen moet nakijken om de gepaste tranformatie te vinden.

2.5 Awaitable methodes

.NET 4.5 introduceert *awaitable* methodes. Deze methodes zijn herkenbaar aan de suffix *Async*. Hun uitvoering zal (normaal gezien :)) door het keyword *await* worden voorafgegaan. In dat geval zal de rest van de code pas worden uitgevoerd nadat de (asynchrone) oproep van de Async-methode is afgelopen (dit wordt meer in detail toegelicht in het hoofdstuk omtrent delegates en het hoofdstuk omtrent backtracking).

```
public static async Task XMLSerialise(Bord brd)
    {
    StorageFolder storageFolder = ApplicationData.Current.LocalFolder;
    StorageFile sf = await storageFolder.CreateFileAsync("numberlegions
    Stream s = await sf.OpenStreamForWriteAsync();
    XmlSerializer serializer = new XmlSerializer(typeof(Bord));
    serializer.Serialize(s, brd);
    s.Dispose();
    }
```

Figuur 2.4: Await demo

lol, dat werkt!

Belangrijk hierbij op te merken zijn volgende elementen:

- indien u een methode *await*, dan kan dit enkel in een methode die als *async* gedefinieerd werd;
- indien een methode als *async* gedefinieerd wordt is ze op haar beurt awaitable.
  - een async methode zal, indien opgeroepen zonder await, asynchroon worden uitgevoerd: de oproepende methode wacht niet op voltooiing vooraleer verder te gaan met zijn logica:

```
        async void brdNumbers_SpelerToggled(object se
            {
            _isToetsVerwerkingBusy = false;
            pSpeler1ScoreA.Text = "" + brdNumbers.Sco
            pSpeler2ScoreA.Text = "" + brdNumbers.Sco
            pSpeler1ScoreB.Text = "" + brdNumbers.Sco
            pSpeler2ScoreB.Text = "" + brdNumbers.Sco
            await Bord.XMLSerialise(brdNumbers);
            await ZetVolgendeZet(e);
            }
```

Figuur 2.5: Await demo async methode

2.6 (XML) serialisatie in .NET

aanbrengen In een context van application life cycle

2.7 Var keyword

als je type kent: gebruik het! var is vooral interessant om het resultaat type van een linq query niet zelf te moeten tikken.

2.8 TryParse methodes

boolean result, twee parameters, tweede parameter is ref, lijkt me zeer toepasbaar bij dataconverters in XAML

2.9 Dynamic keyword

2.9.1 lock en volatile keywords

2.9.2 WPF toolkit

2.9.3 Application settings in WPF

Best way to bind WPF properties to ApplicationSettings in C#?

2.9.4 conditional[debug]

2.9.5 custom build

2.9.6 sys tools

lol, dat werkt!!!
jeDaels

Hoofdstuk 3

Op aanvraag: Data Binding in WPF

Databinding biedt WPF de mogelijkheid om declaratieve visualisaties voor data te definiëren via XAML-code. Het betrachtte voordeel is het aanpasbaar maken van de voorstelling zonder de achterliggende programmeer code te moeten aanpassen. Dit introduceert bijna automatisch de mogelijkheid om verschillende templates op hetzelfde gegeven toe te passen. In het MVVM-model wordt deze manier van werken sterk gepromoot.

3.1 De klasse nMCTDocent

Dit hoofdstuk betreft het visualiseren van data door middel van bindings. Vooraleer we de bindings toelichten moeten we dan ook een dataklasse opzetten:

```
public class nMCTDocent
    {
    public String FNaam { get; set; }
    public String VNaam { get; set; }
    public string Foto { get; set; }

    public static ObservableCollection<nMCTDocent> Docenten
        {
        get
            {
            ObservableCollection<nMCTDocent> l = new ObservableCollection<nMCTDocent>();
            l.Add(new nMCTDocent() { FNaam = "Gates", VNaam = "Bill", Foto = @"fotos\bgates.png" });
            l.Add(new nMCTDocent() { FNaam = "Knuth", VNaam = "Donald", Foto = @"fotos\dknuth.jpg" });
            l.Add(new nMCTDocent() { FNaam = "Turing", VNaam = "Alan", Foto = @"fotos\aturing.png" });
            l.Add(new nMCTDocent() { FNaam = "Lovelace", VNaam = "Ada", Foto = @"fotos\alovelace.png" });

            return l;
            }
        }
```

Figuur 3.1: De klasse nMCTDocent

- het *ObservableCollection* type wordt verder in deze tekst toegelicht. Momenteel volstaat het om dit type te beschouwen als een soort *List*;

- het is belangrijk dat de docentlijst via een property wordt aangeleverd. Op deze manier zullen we wat verder in de tekst lijsten kunnen binden met de *Docenten*, waarbij elke docent op zijn beurt zal gebind worden aan zijn visualisatie.

3.2 DisplayMemberPath: the old way

Databinding bestond reeds voor het .NET tijdperk. Zo kon men de (tekst-) visualisatie van een data- element in een lijst koppelen aan een property- waarde van dit data-element. De koppeling gebeurde door het *DisplayMember* in te stellen op de lijst. In WPF bestaat deze mogelijkheid ook nog en moeten we het *DisplayMemberPath* voor de lijst instellen:

```
<ComboBox x:Name="cboDocenten" DisplayMemberPath="FNaam"
          Grid.Row="1" Grid.Column="2"  Margin="3" />
```

Figuur 3.2: DisplayMemberPath binding

Een foutieve property-naam bij het binden resulteert niet in een compilatie of runtime- fout: het gegeven wordt gewoon niet gevisualiseerd. U kan dit wel terugvinden in de output van uw programma (u merkt dat de binding case sensitive is):

```
System.Windows.Data Error: 40 : BindingExpression path error: 'fNaam' property not found on 'object' ''nMCTDocent'
System.Windows.Data Error: 40 : BindingExpression path error: 'fNaam' property not found on 'object' ''nMCTDocent'
System.Windows.Data Error: 40 : BindingExpression path error: 'fNaam' property not found on 'object' ''nMCTDocent'
System.Windows.Data Error: 40 : BindingExpression path error: 'fNaam' property not found on 'object' ''nMCTDocent'
```

Figuur 3.3: BindingPathError

- in bovenstaande situatie moet de invulling van de combobox elders ($C^{\#}$, de zogeheten *code behind*) gebeuren. Dit is de voornaamste tekortkoming van deze benadering waardoor de visualisatie sterk afhankelijk van het programmeer werk blijft;

- elk element in de combobox wordt ondervraagd om de waarde van zijn *FNaam* property te bekomen (u weet hoe dit kan via reflection);

- de bekomen waarde (een tekst) wordt getoond als voorstelling van het object. Eigenlijk ben ik verrast dat deze binding enkel teksten kan tonen (u zal verder in de tekst merken dat binding zeker niet beperkt is tot tekst). Onderstaande ILSpy informatie lijkt dit voor wat betreft de *DisplayMemberPath*- situatie echter te bevestigen:

```
// System.Windows.Controls.ItemsControl
private void UpdateDisplayMemberTemplateSelector()
{
    string displayMemberPath = this.DisplayMemberPath;
    string itemStringFormat = this.ItemStringFormat;
    if (string.IsNullOrEmpty(displayMemberPath) && string.IsNullOrEmpty(itemStringFormat))
    {
        if (this.ItemTemplateSelector is DisplayMemberTemplateSelector)
        {
            base.ClearValue(ItemsControl.ItemTemplateSelectorProperty);
        }
        return;
    }
    DataTemplateSelector itemTemplateSelector = this.ItemTemplateSelector;
    if (itemTemplateSelector != null && !(itemTemplateSelector is DisplayMemberTemplateSelector) && (base.ReadLo
    {
        throw new InvalidOperationException(SR.Get("DisplayMemberPathAndItemTemplateSelectorDefined"));
    }
    this.ItemTemplateSelector = new DisplayMemberTemplateSelector(this.DisplayMemberPath, this.ItemStringFormat)
}
```

Figuur 3.4: ILSpy DisplayMemberPath info

3.3 DataContext: default binding source

lol, dat werkt!!!

3.3.1 DataContext: duiding

Opdat een visualisatie een dataelement zou kunnen tonen moet het dit dataelement natuurlijk kennen. In een WPF- context spreken we dan van de *DataContext* van het UI-element. Een goed begrip van deze *DataContext* is cruciaal om databinding goed te begrijpen. Volgende punten zijn belangrijk:

- indien een UI-element zijn eigen datacontext niet expliciet instelt wordt de datacontext van de UI-parent overgenomen;
  - dit is de hoeksteen van het *MVVM*- gebeuren: we maken één static resource die we instellen als datacontext van het window, alle UI-elementen in dat window erven deze datacontext. Het is dan natuurlijk nodig dat elk viseel element een property-tegenhanger heeft in de viewmodel klasse;
  - uitzonderingen zijn items in lijsten, waarbij elke visualisatie van een lijst-element dit lijstelement als datacontext krijgt ingesteld (automatisch);
- een datacontext instellen zonder deze te gebruiken in een binding is geen enkel probleem;
- bij het ontwikkelen van mijn eigen templated controls zal ik in de constructor de control (bijna) altijd als zijn eigen datacontext instellen.

3.3.2 Datacontext: instellen op een window

3.3.2.1 ViewModel voor de demo

Ook nu moeten we een klasse (het viewmodel) opzetten om die later te gebruiken in de visualisatie:

```
public class BindingsDemoVM
    {
    public string WindowTitel {
        get { return "Binding demo"; }
        }
    public ObservableCollection<nMCTDocent> Docenten {
        get { return nMCTDocent.Docenten; }
        }
```

Figuur 3.5: ViewModel klasse

In eerste instantie levert het viewmodel twee properties:

- *WindowTitel* die gebind zal worden aan de Title- property van het Window;
- *Docenten* die een lijst van docenten zal aanleveren, die in een lijstcontrol (Combobox, ListView, GridView, ..) getoond kan worden.

Jef Maels
lol, dat werkt!!!

3.3.2.2 Instellen van de Window DataContext

- in XAML moet een element reeds gedefinieerd zijn vooraleer het gebruikt kan worden. Omdat de resources van een Window na de *Window*- tag komen, en de Title- property in die Window- tag wordt ingesteld is het geen optie om de datacontext in de *Window.Resources* te definiëren. De plaats om de DataContext van het opstartwindow te definiëren is in de resources van de App.xaml- file (deze wordt gelezen voor het Window):

```
<Application.Resources>
    <src:BindingsDemoVM x:Key="vm" />
</Application.Resources>
```

Figuur 3.6: Application Resources

- op basis van de key kan dit (static) resource gebruikt worden in een binding:

```
<Window x:Class="BindingsDemo.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/pr
    Height="350" Width="525"
    Title="{Binding Path=WindowTitel}"
    DataContext="{Binding Source={StaticResource vm}}"
    >
```

Figuur 3.7: Window binding

De window- tag bevat twee bindings:

- het binden van de DataContext aan de static resource *vm*;
  - indien gewenst kan de DataContext ook via code in de constructor worden ingesteld. Dit kan zinnig zijn indien u geen DataContext instantie als resource kan aanmaken;
- het binden van de *Title* property van het window aan de *WindowTitel* property van de datacontext (indien geen source wordt ingesteld op de binding, dan is de datacontext de source);
  - indien de DataContext van de controls in het Window niet wordt overschreven kan nu elke String- property van elke control in het window (ook deze die elders in een template worden beschreven en nog geladen moeten worden) gebind worden aan de *WindowTitel* van de overgenomen datacontext (bemerk dat de datacontext niet expliciet wordt ingesteld):

```
<TextBlock Text="{Binding WindowTitel}" Grid.Row="3" Grid.Column="1" />
```

Figuur 3.8: DataContext binding

l, dat werkt!!

3.3.2.3 Binden van de ItemsControl.ItemsSource Property

Indien we lijsten wensen te visualiseren zullen we hiervoor wellicht de ItemsControl klasse (of een afgeleide) gebruiken. Deze klasse heeft een *ItemsSource* property van het (Interface-) type IEnumerable: dit kan zowel een List, een Array, .. zijn, zolang de IEnumerable interface maar geïmplementeerd wordt:

ItemsControl.ItemsSource Property

.NET Framework 4.5 | Other Versions | 0 out of 5 rated this helpful

Gets or sets a collection used to generate the content of the ItemsControl.

Namespace: System.Windows.Controls
Assembly: PresentationFramework (in PresentationFramework.dll)
XMLNS for XAML: http://schemas.microsoft.com/winfx/2006/xaml/presentation, http://schemas.

Syntax

C# | C++ | F# | VB

```
[BindableAttribute(true)]
public IEnumerable ItemsSource { get; set; }
```

Figuur 3.9: ItemsSource declaratie

Indien u in een *C#*- window *F12* drukt op het *ObservableCollection*- type wordt de gekende code getoond:

```
public class ObservableCollection<T> : Collection<T>, INotifyCollectionChanged, INotifyPropertyChanged
```

Figuur 3.10: Observable hoofding

Indien we nu terug *F12* duwen op *Collection* dan bekomen we:

```
public class Collection<T> : IList<T>, ICollection<T>, IList, ICollection, IReadOnlyList<T>, IReadOnlyCollection<T>, IEnumerable<T>, IE
```

Figuur 3.11: Collection hoofding

Hieruit kunnen we afleiden dat de *ObservableCollection*- klasse, via het erven van de *Collection*- klasse ook de *IEnumerable* interface implementeert, en de *Docenten* property van het modelview dus geschikt is als bindings source van de ItemsSource van een ComboBox:

```
<ComboBox x:Name="cboDocentenBound" Grid.Row="2" Grid.Column="2"
          DisplayMemberPath="FNaam" Margin="3"
          ItemsSource="{Binding Docenten}" />
```

Figuur 3.12: ItemsSource binding

Opdat de combobox zou weten welke informatie van het dataelement getoond moet worden wordt ook nu terug de *DisplayMemberPath* property ingesteld.

Jef Rycke
lol, dat werkt!!

3.3.2.4 Instellen van de ItemsControl.ItemTemplate Property

Om een rijke user experience mogelijk te maken willen we natuurlijk meer dan enkel tekst waardes kunnen tonen. Indien u een DataTemplate hebt voor uw te tonen dataelement kan u dit in de *ItemsControl* instellen met behulp van de *ItemTemplate* property die dan wordt toegepast op elk van de items in de *ItemsSource*:

```
<ComboBox x:Name="cboDocentenBoundTemplated" Grid.Row="3" Grid.Column="2"
          Margin="3"
          ItemTemplate="{StaticResource tmplDocenten}"
          ItemsSource="{Binding Docenten}" />
```

Figuur 3.13: ItemTemplate instellen

Bovenstaande code veronderstelt het bestaan van de *tmplDocenten* datatemplate. Deze wordt gedefinieerd in een resource sectie van bijvoorbeeld het window of de app.xaml file:

```
<DataTemplate x:Key="tmplDocenten" DataType="src:nMCTDocent" >
    <Grid Margin="4">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" SharedSizeGroup="Key" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <TextBlock Text="{Binding FNaam}" FontWeight="Bold"  />
        <Image Source="{Binding Foto}" Grid.Column="1" />
    </Grid>
</DataTemplate>
```

Figuur 3.14: nMCTDocent datatemplate

Het resultaat van deze binding met deze template ziet er als volgt uit:

Figuur 3.15: nMCTDocenten in een ComboBox

3.4 Niet DataContext binding sources

De DataContext, zoals de naam reeds suggereert, laat toe om in een binding de gegevens van het te visualiseren element (de data) te gebruiken. Deze datacontext is niet altijd de bron van het nodige gegeven. Voorbeelden waar de nodige informatie elders gehaald moet worden zijn:

- in de ccCaption control worden images getoond. We wensen deze als vierkant te tonen, wiens breedte gelijk is aan de hoogte van de Grid waarin ze worden geplaatst;
- bij het ontwikkelen van een custom control wensen we de properties die de designer instelt op de control te gebruiken in de template;

Om bovenstaande gevallen te kunnen afhandelen moet de *binding source* ingesteld worden. Dit kan gebeuren aan de hand van één van drie properties:

- Source: indien u een bepaalde property wenst te binden aan een ander (static) resource dan de huidige datacontext, zonder de datacontext aan te passen

- RelativeSource (MSDN): om de binding source aan te geven relatief ten opzichte van de control die de binding uitvoert. De syntax voor deze source instelling steunt op RelativeSourceMode Enumeration. Googlen zal ook hier uw beste vriendje zijn (lees zeker ook de MSDN entry);
- ElementName: indien u de binding source die u wenst te gebruiken bij naam kan identificeren in de Visual Tree.

Wanneer bijvoorbeeld de breedte van een GridColumn gebind moet worden aan de hoogte van die Grid kan onderstaande XAML-code gebruikt worden:

```
<DataTemplate x:Key="tmplDocenten2" DataType="src:nMCTDocent" >
    <Grid Margin="4" Height="60">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" SharedSizeGroup="Key" />
            <ColumnDefinition
                Width="{Binding RelativeSource={RelativeSource FindAncestor,
                AncestorType={x:Type Grid}}, Path=ActualHeight}"/>
        </Grid.ColumnDefinitions>
        <TextBlock Text="{Binding FNaam}" FontWeight="Bold" Style="{StaticResource txbstyle}"  />
        <Image Source="{Binding Foto}" Grid.Column="1" />
    </Grid>
</DataTemplate>
```

Figuur 3.16: RelativeSource binding

lol, dat jeDaet

3.5 TypeConverter

Zie TypeConverter

Hoofdstuk 4

Delegates

4.1 Delegates: duiding

In een volledig object georiënteerde programmeeromgeving is het mogelijk om code- onderdelen als objecten te aanzien. Dit betekent dat deze onderdelen (routines, classes, assemblies, types, ..) instanties zijn van klasses die dergelijke code elementen beschrijven.

Routine- objecten (het onderwerp van dit hoofdstuk) zijn instanties van een *Delegate-type*. Deze Delegate- types zijn klasses die erven van de *Delegate*- class. Deze laatste is zelf geen Delgate-type (er zijn geen routines die instanties zijn van de basis Delegate-class).

In het hoofdstuk omtrent reflection zullen we ook andere code onderdelen als objecten verwerken. Routines zullen daar ook als instanties van (Reflection-) classes terugkeren. In een reflection context zullen deze instanties vooral beschrijvende (meta-) informatie bevatten.

Delegates spelen in .NET een belangrijke rol bij het implementeren van events, multithreading en codeplumbing. Ook de implementatie van WCF en (attached) dependency properties maken gebruik van delegates. Delegates zijn dan ook zonder twijfel een belangrijke werkstuk voor de betere programmeur!

lol, dat werkt!!!

4.1.1 Relevante namespace

- *System*

4.2 Delegates: technische intro

4.2.1 Demo-code en toelichting

De meest voor de hand liggende actie die we op een routine willen toepassen, is deze uitvoeren. Hierbij is het van belang te weten welke argumenten moeten voorzien worden en van welke type het eventuele resultaat een instantie zal zijn. Een delegate- type definitie zal zich hiertoe beperken: welke zijn de argumenttypes, en welk is het eventuele resultaat type? De naam van de routine en de parameters, en de geïmplementeerde logica, zijn volledig irrelevant voor de delegate type definitie. Een voorbeeld delegate- type definitie:

```
public delegate void DoeIetsMetTekstDelegate(string sTekst);
```

Figuur 4.1: Delegate definitie

Merk hierbij volgende punten op:

- het keyword *delegate* heeft aan dat we een delegate-type definiëren (een speciale klasse, zonder body);
- het return type wordt genoteerd voor de naam van het delegate type. In het voorbeeld is dit return type void (we verwachten geen resultaat);
- de argumentnamen zijn niet van belang (en moeten zeker niet overeenstemmen met de argumentnamen gebruikt in de routines die instanties van deze klasse zullen zijn). De argumenttypes zijn *wel* belangrijk!

Jolldatwerkt!!!

```
public delegate void doeIetsMetTekst(String tekst);

public partial class MainWindow : Window
    {
    private void btnDemoUCLC_Click(object sender, RoutedEventArgs e)
        {
        doeIetsMetTekst routine = new doeIetsMetTekst(toonUC);  //maak een routinereferentie en koppel ze
        routine.Invoke("hAllO");    //start de methode gekoppeld aan deze routinereferentie, en voorzie de oproep van de nodige parameters
        routine = new doeIetsMetTekst(toonLC);
        routine.Invoke("hAllO");
        }
    private void toonUC(string tekst) { MessageBox.Show(tekst.ToUpper()); }
    private void toonLC(string tekst) { MessageBox.Show(tekst.ToLower()); }
```

Figuur 4.2: Delegate demo: uppercase en lowercase

Bovenstaande demo-code zal u wellicht niet overtuigen van het belang van delegates: het aanroepen van een methode wordt gewoon complexer gemaakt. Verder in dit hoofdstuk behandelen we een aantal (niet exhaustief (=uitputtend)) situaties waarbij het gebruik van delegates een elegante oplossing biedt.

Een delegate- definitie is niet hetzelfde als een signatuur definitie: de routinenaam maakt onderdeel uit van de signatuur (en niet van het delegate- type), en het returntype is geen onderdeel van de signatuur, maar wel van belang in de delegate definitie.

4.2.2 Covariance en contravariance

Wat indien een routine hoofding niet volledig overeenstemt met de delegate definitie? Een routine kan op twee manieren afwijken van een delegate type definitie, en toch nog als instantie van dit delegate type worden aanzien: covariance en contravariance. Beide worden beschreven in Covariance and Contravariance in Delegates (C# Programming Guide). Een combinatie van beide is natuurlijk ook toegelaten.

4.2.2.1 Covariance

Indien het returntype van een routine een subclasse is van het return type van het delegate type, dan is deze routine toch een instantie van dit delegate type. Uiteraard levert dit geen problemen op, gezien elk resultaat kan gecast worden naar het return type van het delegate type.

4.2.2.2 Contravariance

Indien een argument van de routine minder gespecialisserd is dan het overeenkomstige argument in het delegate type is er uiteraard ook geen probleem: elke actuele parameter moet voldoen aan het delegate (formele) parameter type, en voldoet dus zeker ook aan het algemenere routine parameter type.

lol, dat werkt!!!

4.3 Delegates en events

Events leveren een mechanisme waarbij een server-object (levert diensten) initiatief neemt om code door een client-object te laten uitvoeren. In .NET wordt dit gerealiseerd door middel van delegates (dit in tegenstelling tot JAVA waar men gebruikt maakt van interfaces (denk bijvoorbeeld aan de *ActionListener* interface)).

Het client of server zijn van een object ten opzichte van elkaar hangt af van de te leveren dienst: indien we een Button op een Window plaatsen dan is deze Button client van het Window omwille van de visuele ondersteuning (het Window levert een dienst), en is het Window evenzeer client van de Button omdat de Button onder andere een click- dienst aanlevert. Het is dus zeker niet zo dat een client/server relatie gelezen moet worden als een klein/groot- relatie.

4.3.1 Definitie van een event

De definitie van een event voorziet een delegate- typed field van het keyword *event*:

```
public delegate void EventVoorbeeldNietVolgensGuidelines(string tekst); //delegate voor het event
public class DemoTekstVak : TextBox
    {
    public event EventVoorbeeldNietVolgensGuidelines TekstIsVijfLang; //eventdefinitie
    public DemoTekstVak()
        {
```

Figuur 4.3: Event definitie (zonder guidelines)

De delegate in de eventdefinitie beschrijft de client-routines die aan dit event gekoppeld kunnen worden. Bemerk dat de code in dit voorbeeld *niet* voldoet aan de guidelines omtrent eventhandling.

Bovenstaande code is onderdeel van een gespecialiseerde TextBox klasse die voorzien wordt van een TekstIsVijfLang event. Het is de bedoeling dit event af te vuren indien de tekst in het tekstvak exact 5 tekens lang wordt.

Jef Dae/s
dat werkt!!!

4.3.2 Abonneren (to subscribe) op een event

De syntax nodig om een client te abonneren op een event is verschillend met deze van VB.NET (waarbij een *handles* clause achteraan een routine werd toegevoegd). In $C^{\#}$wordt een routine gekoppeld aan een event door middel van de *+=*- operator:

```
public MainWindow()
    {
    InitializeComponent();
    txt5Tekens.TekstIsVijfLang += new EventVoorbeeldNietVolgensGuidelines(txt5Tekens_TekstIsVijfLang);
    }
void txt5Tekens_TekstIsVijfLang(string tekst)
    {
    Console.WriteLine(tekst + " is 5 tekens lang");
    }
```

Figuur 4.4: Event abonnering

Visual Studio ondersteunt het abonneren op volgende wijze:

- indien u na de *+=*- operator de *TAB* toets indrukt wordt automatisch een correcte right-hand expressie gegenereerd;
- indien u hierna nog eens *TAB* indrukt zal ook een correcte eventhandler routine gegenereerd worden, rekening houdend met de event definiërende delegate.
- indien u dubbelklikt op een control wordt er een *+=*- expressie aangemaakt in de generated code en wordt een passende eventhandler in de *.cs*- fiel geplaatst;

4.3.2.1 Event subscription: verschil met VB.NET

- Handles- clause: Hiervoor bestaat geen $C^{\#}$- equivalent: $C^{\#}$kent enkel de *+=*- operator om routines aan events te koppelen;

- AddHandler: De VB.NET handles clause is niet in staat om routines *at run-time* aan events te koppelen. Hierdoor is deze werkwijze ongeschikt om events van bijvoorbeeld dynamisch toegevoegde controls op te vangen. VB.NET kent de *AddHandler* instructie om at run-time routines aan events te koppelen. Indien u de documentatie erop naleest merkt u dat VB.NET de *AddressOf*- operator gebruikt in combinatie met de AddHandler instructie.

4.3.3 Afvuren van een event

De klasse die het event definieert kan dit event ook uitvoeren. Het is belangrijk om te testen of er clients zijn voor dit event (indien er geen clients zijn is het field *null* en zou het afvuren een uitvoeringsfout veroorzaken):

rkt!!

```
public delegate void EventVoorbeeldNietVolgensGuidelines(string tekst); //delegate voor het event
public class DemoTekstVak : TextBox
    {
    public event EventVoorbeeldNietVolgensGuidelines TekstIsVijfLang; //eventdefinitie
    public DemoTekstVak()
        {
        this.TextChanged += new TextChangedEventHandler(DemoTekstVak_TextChanged);
        }

    void DemoTekstVak_TextChanged(object sender, TextChangedEventArgs e)
        {
        if (this.Text.Length == 5)
            if (TekstIsVijfLang != null)    //minstens 1 client voor dit event nodig, anders null
                TekstIsVijfLang(this.Text); //starten van het event (voor alle clients)
        }
    }
```

Figuur 4.5: Event afvuren (zonder guidelines)

Bemerk dat in bovenstaande code, die het afvuren van een event illustreert, er ook een event abonnering aanwezig is: het object abonneert zichzelf (in de constructor) op zijn eigen *TextChanged*- event (door middel van de *+=- operator*) om na te kijken of de tekst na wijziging exact vijf tekens lang is.

4.3.3.1 Afvuren van een event: verschil met VB.NET

- *RaiseEvent* is de instructie om in VB.NET een event af te vuren. Het is *niet* nodig om te controleren of er clients zijn voor dit event.

4.3.4 Meerdere clients voor hetzelfde event

Het is perfect mogelijk om meerdere event-handlers te koppelen aan eenzelfde event. Bij het afvuren van het event zullen de gekoppelde routines in volgorde, één voor één uitgevoerd worden.

4.4 Event Guidelines!

Hoewel er technisch geen beperkingen worden opgelegd aan de event definiërende delegates worden door Microsoft volgende guidelines beschreven:

- het return type is *void*;
- de delegate heeft exact twee argumenten:
  - Object sender: het object waardoor het event wordt gestart;
  - EventArgs e: EventArgs mag vervangen worden door een klasse die overerft van EventArgs indien u extra informatie aan het event wenst toe te voegen (voorbeelden hiervan zijn de mouselistener events);
- het afvuren van het event gebeurt in een overschrijfbare (*protected virtual*) methode. Dit laat ervende klasses toe om het event gedrag aan te passen (zie DemoButton in het demo project). De methodenaam is *onEvent* waarbij Event natuurlijk wordt vervangen door de naam van het event.
- gebruik *System.EventHandler*< *T*> in plaats van zelf aangemaakte delegate definities.

Deze guidelines *moeten* in deze cursus worden toegepast opdat uw event-implementaties als correct zouden worden aanzien.

Als een eerste oefening kan men de event implementatie, zoals hierboven beschreven, hercoderen zodat ze voldoet aan de guidelines.

Wie niet (voldoende) bekend is met *generics* (*System.EventHandler*< *T*>) wordt verondersteld dit op te zoeken. Indien zelfstudie onvoldoende duidelijkheid schept kan u vragen stellen in de komende lessen.

lol, dat werkt!!!

4.5 Omtrent Events:

4.5.1 Static events

Meestal associëren we een event met een instantie. Om te abonneren op een event van een instantie hebben we een referentie naar die instantie nodig. Dit is niet altijd mogelijk: veronderstel dat client code wenst te reageren op het aanmaken van een nieuw object van een klasse: het is niet mogelijk in te tekenen op het *PersoonCreated* event van een nog onbestaand object. .NET voorziet de mogelijkheid om static events te definiëren: de client abonneert dan op een (static) event van een klasse:

```
public class clsPersoon
    {
    public static event EventHandler<PersoonCreatedEventArgs> PersoonCreated;
    public string FNaam { get; set; }   //property definitie
    public string VNaam { get; set; }   //property definitie
    public clsPersoon() : this("","")
        {
        }
    public clsPersoon(string fnaam, string vnaam)
        {
        FNaam = fnaam;
        VNaam = vnaam;
        if (PersoonCreated != null)
            PersoonCreated(null, new PersoonCreatedEventArgs(this));
        }
    public override string ToString()
        {
        return this.FNaam + ", " + this.VNaam;
        }
    }
public class PersoonCreatedEventArgs : EventArgs
    {
    private clsPersoon _persoon;
    public PersoonCreatedEventArgs(clsPersoon persoon) { _persoon = persoon; }
    public clsPersoon persoon { get { return _persoon; } }
    }
```

Figuur 4.6: Static event definitie (volgens de guidelines)

```
public MainWindow()
    {
    InitializeComponent();
    clsPersoon.PersoonCreated += new EventHandler<PersoonCreatedEventArgs>(clsPersoon_PersoonCreated);
    }

void clsPersoon_PersoonCreated(object sender, PersoonCreatedEventArgs e)
    {
    Console.WriteLine(e.persoon.ToString());
    }
```

Figuur 4.7: Static event client

4.5.2 Delegates of Interfaces: .NET of JAVA events

In .NET gebruikt men delegates om eventhandling te realiseren. In JAVA definieert men interfaces om eventhandling mogelijk te maken: een instantie van een interface implementerende klasse kan dan als eventhandler aan een object toegekend worden. Welke is te verkiezen?

Bij manuele codering (wie doet dat nog?) moet in JAVA elk interface element voorzien worden, ook al bent u enkel geïnteresseerd in een specifieke routine. In .NET moet u enkel die events die u interesseren van code voorzien. Beide oplossingen zijn voldoende eenvoudig om een intuitief gebruik mogelijk te maken (zeker indien de IDE u ook nog wat helpt). Beide benaderingen zijn dan ook sterk gelijkwaardig: ik verkies er geeneen boven de andere.

4.5.3 Routed events

Voorgaande eventhandling is reeds van bij de start van .NET aanwezig in het framework. Recentelijk (WPF, .NET 3.5) werden ook *routed events* geïntroduceerd. Deze events zijn WPF specifiek en worden geïntroduceerd om volgende problemen op te lossen:

- veronderstel een Button, met daarop een tekening (Image);
- de gebruiker klikt op de tekening: wellicht wil hij de button inklikken? Misschien ook niet? Hoe wordt de Button op de hoogte gesteld van het klikken op de Image? Wat indien er geen Image is?

Een routed event is een event (anders dan voorgaande) om binnen een geneste lijst van controls een gebeurtenis door te geven. We spreken van *tunneling events* indien de container het event doorgeeft naar zijn child, en van *bubbling events* indien het child het event doorgeeft aan zijn container. Een parent (bijvoorbeeld Window)is in staat om te subscriben (addHandler) op het routed event (bijvoorbeeld *TextChanged*- event van een bij abonnering niet gekend child (zie ook Sudoku routed events).

lol, dat werkt!!!

Indien een WPF- gebeurtenis zich voordoet zal het doorgeven van events in twee stappen gebeuren:

- eerst wordt het event van de containers naar de children doorgegeven: *tunneling*. Deze events worden prefixed met *preview*;
- daarna volgt een bubbling fase waarbij de gewone event naam wordt gebruikt;
- indien een control meent het event correct af te handelen kan de handled property van het tweede event-argument op *true* worden geplaatst, wat het doorgeven van die specifieke fase (grotendeels) stopt.

Interessant om weten:

- routed events bestaan enkel in een WPF-omgeving. Ze kunnen dan ook niet gebruikt worden om eventhandling buiten de WPF omgeving (uw visualisatie) te implementeren.
- snoop is een leuke tool om WPF-constructies te onderzoeken. Ook het verloop van routed events kan met deze tool worden nageplozen.

4.6 Delegates en multithreading

4.6.1 Multithreading: wat bedoelt u?

Bij het uitvoeren van een programma worden instructies één voor één door de CPU verwerkt. Een thread is een onderdeel van een proces (programma) dat in staat is om opeenvolgende instructies uit te voeren. Een multithreaded programma heeft meerdere onafhankelijk van elkaar uitvoerende threads. Wanneer het programma wordt uitgevoerd door een multithreaded operating system worden verschillende zaken op hetzelfde moment uitgevoerd.

4.6.2 Multithreading: iets voor ons?

Multithreaded programmeren is *ongeloofelijk* delicaat: indien gelijktijdige threads eenzelfde variabele wensen aan te passen krijgen we de problemen die we kennen vanuit de cursus databases: *lost update*, *inconsistent read* en *ghost data* om er slechts enkele te noemen. Het schrijven van correcte multithreaded toepassigen is dan ook geen *walk in the park/piece of cake*.

lol, dat werkt!!!

Betekent dit dat we geen multithreading kunnen programmeren? Natuurlijk niet: het ophalen van een grote lijst gegevens, het verwerken van een lijst gegevens die niet gewijzigd worden: allemaal ideale kandidaten om multithreaded uitgewerkt te worden. Indien er zich concurrency (gelijktijdigheid) problemen kunnen voordoen zullen we locks gebruiken. Bij het database programmeren worden locks door het RDBMS geplaatst. In $C^{\#}$(en VB.NET) kunnen we dit zelf programmeren met behulp van de *lock*- instructie.

Wanneer we spreken over multithreaded toepassingen hebben we het veelal ook over:

- parallel processing: parallelle (gelijktijdige) uitvoeringen van meerdere threads impliceren multithreading;
- asynchrone verwerking (in tegenstelling tot synchrone verwerking bij single threaded toepassingen): hierbij wordt een routine gestart en wachten we niet op het einde van deze routine vooraleer we verder gaan. Asynchrone verwerking wordt in .NET gerealiseerd met behulp van delegates. Een voor de hand liggend probleem dat moet worden opgelost: hoe weet mijn programma dat een asynchrone uitvoering afgelopen is (om bijvoorbeeld een *printen beëindigd* boodschap te kunnen tonen?

Multithreaded applicaties zijn ook op een single- core CPU belangrijk: zo kan men het printen van een document op de achtergrond laten uitvoeren, terwijl de user-responsiveness niet daalt. Met de introductie van multi-core CPU's wordt het ontwikkelen van multi-threaded toepassingen nog interessanter: een single-threaded toepassing zal op een quad core nooit meer dan vijventwintig procent van de beschikbare rekenkracht kunnen gebruiken. Dit maakt het multi-threaded zijn van uw toepassing belangrijk indien u het volle potentieel van de machine wenst te gebruiken.

4.6.3 Asynchrone processing en delegates

4.6.3.1 Asynchroon, geen AsyncCallback

Bij een synchrone verwerking zal een routine de volledige utivoering van een instructie (eventueel een deelprobleem) afwachten vooraleer de volgende instructie wordt gestart. Zo zal de uitvoering van btnDemoUCLC_click in Delegate demo: uppercase en lowercase na het tonen van de eerste messagebox wachten tot de gebruiker op *ok* klikt vooraleer de tweede messagebox wordt getoond.

Wanneer we de code aanpassen door de *invoke/1*- methods van de delegates te vervangen door *BeginInvoke/3* dan worden de delegate routines asynchroon ten opzichte van de oproepende methode uitgevoerd:

```
private void btnAsyncDemo_Click(object sender, RoutedEventArgs e)
    {
    doeIetsMetTekst routine = new doeIetsMetTekst(toonUC);  //maak een routinereferentie en koppel ze
    routine.BeginInvoke("hAllO", null, null);    //start de delegate asynchroon
    routine = new doeIetsMetTekst(toonLC);
    routine.BeginInvoke("hAllO",null,null);
    }
```

Figuur 4.8: Asynchrone verwerking demo

Jef Daels lol, dat

Op te merken bij de nieuwe situatie:

- deze uitvoering resulteert in drie threads: de thread die het klik- event afhandelt, en een tread voor elk van de asynchrone oproepen;

- *BeginInvoke/3* kent drie parameters in plaats van één: de twee laatste parameters van een asynchrone delegate oproep bevaten een callback routine (de routine die wordt uitgevoerd als de asynchroon gestartte routine eindigt) en een status object als laatste argument (beide komen in het volgende punt aan bod). Het is de bedoeling om in dit status object informatie, verzameld gedurende de asynchrone uitvoering (bijvoorbeeld het goed of slecht aflopen) terug te geven aan de oproeper. Dit laatste argument is (later) terug te vinden als state property van de *IAsyncResult*- parameter van de *AsyncCallback* routine.

- de callback methode wordt via een AsyncCallBack- delegate aan de asynchrone oproep meegegeven (de callback methode is die routine die zal worden uitgevoerd wanneer de asynchrone oproep afgelopen is). Het is belangrijk op te merken dat de callback routine wordt uitgevoerd op de asynchrone thread!

4.6.3.2 Asynchroon, met AsyncCallback, zonder UI

```
private void btnAsyncDemo_Click(object sender, RoutedEventArgs e)
    {
    doIetsMetTekst routine = new doIetsMetTekst(toonUC);  //maak een routinereferentie en koppel ze
    routine.BeginInvoke("hAllO", new AsyncCallback(asyncAfgelopen), this);    //start de delegate asynchroon
    routine = new doIetsMetTekst(toonLC);
    routine.BeginInvoke("hAllO",null, null);
    }
private void asyncAfgelopen(IAsyncResult iar)
    {
    toonResultaat(iar);
    }
private void toonResultaat(IAsyncResult iar)
    {
    object state = iar.AsyncState;  //bevat laatste argument van de oproep
    this.Title = "afgelopen";
    }
```

Figuur 4.9: AsyncCallback demo

In bovenstaande figuur zal de eerste asynchrone uitvoering bij het aflopen de routine *asyncAfgelopen* uitvoeren (op de nieuwe thread). In het *IAsyncResult* argument vinden we informatie omtrent de uitgevoerde routine. De callback routine wordt uitgevoerd op de nieuw gestarte thread. Bovenstaande stukje code loopt fout omdat we op de nieuwe thread de UI wensen aan te passen:

dat werkt!!

Figuur 4.10: UI-thread problem

In een Windows-omgeving is het enkel mogelijk om de UI-elementen aan te passen op de thread die deze UI-elementen aanmaakte. Indien we in de UI feedback omtrent een afgelopen asynchrone methode wensen te zien, zullen we in staat moeten zijn om naar de UI-thread terug te keren. U raadt het al: dit gebeurt met behulp van delegates.

4.6.3.3 Asynchroon, met AsyncCallback, met UI

```
private void btnAsyncDemo_Click(object sender, RoutedEventArgs e)
    {
    doeIetsMetTekst routine = new doeIetsMetTekst(toonUC);  //maak een routinereferentie en koppel ze
    routine.BeginInvoke("hAllO", new AsyncCallback(asyncAfgelopen), this);    //start de delegate asynchroon
    routine = new doeIetsMetTekst(toonLC);
    routine.BeginInvoke("hAllO",null, null);
    }
private void asyncAfgelopen(IAsyncResult iar)
    {
    Dispatcher.BeginInvoke(new AsyncCallback(toonResultaat),iar);   //Terug naar de ui thread
    }
private void toonResultaat(IAsyncResult iar)
    {
    object state = iar.AsyncState;  //bevat laatste argument van de oproep
    this.Title = "afgelopen";
    }
```

Figuur 4.11: Dispatcher demo

lol, dat werkt!!!

Het terugkeren naar de UI-thread gebeurt afhankelijk van de UI-technologie:

- WPF: elke visueel element heeft een *Dispatcher* property. Deze kan een thread starten op zijn UI-thread door middel van de *BeginInvoke/2* methode. Hoewel deze methode dezelfde naam draagt als deze die de delegate routines asynchroon start is het een andere methode (geen twee extra argumenten bijvoorbeeld);
- WinForms: elke control heeft een *BeginInvoke* methode. Deze start een delegate op de thread die de control maakte (de UI-thread).

Het is interessant op te merken dat een *Messagebox.show* oproep zichzelf terug op de gepaste thread plaatst.

4.6.4 Multithreading met de Thread klasse

Een alternatief om multithreaded toepassingen te maken kan gebruik maken van de *Thread*-klasse. Dit valt buiten de scope van deze les.

4.6.5 Event based Ansynchronous Pattern (EAP)

Het Event based Asynchronous Pattern plaatst het asynchroon ontwikkelen in de alomgekende event- context:

- zowel de langlopende taak (op de andere thread) als de callback worden gekoppeld aan events (zie het BackgroundWorker voorbeeld hieronder);
- op basis van de uitleg omtrent events zou het duidelijk moeten zijn dat dit een wrapper-oplossing rond de asynchroon uit te voeren delegate en zijn callback is.
- het pattern voorziet ook de mogelijkheid om een langdurende bewerking te onderbreken en om voortgangs events af te vuren. U vindt dit terug in de documentatie :).

De BackgroundWorker class is een mooi voorbeeld van deze benadering:

- het DoWork- event: ik kan het niet beter uitleggen dan wat u in MSDN vindt. Bemerk de relatie met RunWorkerAsync
- het RunWorkerCompleted- event wordt ook netjes uitgelegd op MSDN.

4.6.6 Asynchroon programmeren made easy

Voorgaande code, waarbij het nodig was om *Callback delegates* en *DispatcherThreads* te begrijpen maakte het niet eenvoudig om multithreaded toepassingen op te zetten. In .NET 4.5 werd gepoogd om dit (syntactisch) een flink stuk te vereenvoudigen met de introductie van de *await*- en *async* keywords:

dat werkt!!!

```
public async Task XMLSerialise()
    {
    //dit wordt enkel gebruikt als er nog geen bord geserialiseerd was (su
    StorageFolder storageFolder = ApplicationData.Current.LocalFolder; //a
    StorageFile sf = await storageFolder.CreateFileAsync("numberlegionsspl
    Stream s = await sf.OpenStreamForWriteAsync();
    XmlSerializer serializer = new XmlSerializer(typeof(SplashPage));

    serializer.Serialize(s, this);
    s.Dispose();
    }
```

Figuur 4.12: Await- keyword

- het *await*- keyword impliceert volgende zaken:
  - de oproep rechts van het *await*- keyword wordt asynchroon uitgevoerd;
  - de rest van de methode wordt slechts uitgevoerd nadat de asynchrone oproep eindigt. De compiler zal hiertoe de rest van de methode in een delegate wrappen en als callback met de asynchrone oproep meegeven;

- het *async*- keyword in een routine hoofding heeft aan dat deze routine *awaited* kan opgeroepen worden. Meer nog, indien een routine een *awaited*- oproep bevat (typisch een methode oproep die lang kan duren) moet deze routine verplicht *async* gedefinieerd worden. Dit maakt het mogelijk (niet noodzakelijk) om deze routine zelf ook terug *awaited* op te roepen;
  - indien u een *async* methode *niet* awaited oproept dan wordt ze synchroon uitgevoerd (en blokkeert ze dus de oproepende thread).
  - async methodes moeten een return type *Task* (indien geen resultaat- waarde) of *Task<resultaattype>* indien een resultaat wordt terug gegeven.
- bij het ontwikkelen van *Modern Style* apps (Microsoft tablet toepassingen) bent u verplicht deze keywords te gebruiken: het manipuleren van files, het raadplegen van webservices: al deze functionaliteit is enkel nog in een asynchrone versie beschikbaar;
- wat u momenteel nog ontbreekt is de kennis om zelf een asyn methode te ontwikkelen (zonder gebruik te maken van een andere async methode die u await). Dit zal aan bod komen in het hoofdstuk omtrent backtracking waar we onze artificiële intelligentie asynchroon zullen oproepen.
  - voor wie hierop niet kan wachten: we zullen gebruik maken van de TaskCompletionSource klasse.

lol, dat werkt!!
Jef Desmedt

4.6.7 CancellationToken

De CancellationToken structure wordt gebruikt in GeolocationCS sample. Dit token dat bij het starten van een Task wordt meegegeven geeft de caller de mogelijkheid om de Task af te breken. Hiertoe is het nodig dat de gestarte Task regelmatig dit token raadpleegt en indien nodig zijn bewerkingen afbreekt.

4.6.8 Thread.Sleep

Indien meerdere threads met dezelfde prioriteit uitgevoerd moeten worden kan het interessant zijn om de uitvoerende thread te onderbreken zodat de andere thread met dezelfde prioriteit aan bod kan komen. Hiertoe kan u gebruik maken van de (statische) Thread.Sleep - methode. U zal dit nodig hebben om de multithreaded teller oefening op te lossen.

4.6.9 Meer info?

Over multithreading zijn boeken, wellicht ganse bibliotheken, geschreven. De laatste jaren is er hernieuwde aandacht voor deze topic omwille van de multi core processoren (de programmeer talen worden aangepast om parallelle verwerking gemakkelijker te ondersteunen). Een eventueel startpunt om zich in deze materie te verdiepen kan je vinden op Asynchronous Programming Overview. Deze link is wellicht niet beter dan uw eigen favoriete instap link.

4.7 Code plumbing, aka 'the strategy pattern'

Deze laatste topic behandelt het 'at run time' vervangen van code in *één welbepaald* object. Dit is helemaal anders dan overerving: bij overerving zullen we code at compile time aanpassen voor *elke* instantie van de ervende klasse.

4.7.1 Codeplumbing: een XMLViewer

Veronderstel dat u betrokken bent bij de ontwikkeling van een algemene XML-viewer. Deze is in staat om elke mogelijke XML-file in een treeview te visualiseren. Omdat u op voorhand niet alle XML-tags van uw potentële klanten kent kan u in uw XMLviewer geen tag- afhankelijke manipulaties coderen: u weet niet op voorhand hoe een XML-file met CD-gegevens vertoont moet worden. Het is onze doelstelling een XMLViewer control te ontwikkelen waarin de gebruiker bepaalde routines kan vervangen door zijn eigen implementaties. Wij voorzien default implementaties die een basis functionaliteit aanbieden zodat de XML-viewer op zijn minst een standaard gedrag vertoont. We gaan hiervoor als volgt te werk:

lol, dat werkt!!!

- maak een XMLViewer usercontrol die erft van TreeView. Deze overloopt bij het verwerken van een XML-file recursief de verschillende tags en toont deze. Wellicht is het interessant om ook een eigen TreeViewItem klasse te maken. Houd rekening met volgende elementen:

  - plaats de berekening van de getoonde tekst voor elke XML-node in een aparte routine (met de XML-node als argument);
  - plaats de berekening van de kleur voor elke XML-node in een aparte routine (met de XML-node als argument);
  - plaats de test die bepaalt of subnodes moeten verwerkt worden in een aparte routine (met de XML-node als argument);

  U merkt dat elke functionaliteit die we later voor de gebruiker aanpasbaar willen stellen in aparte routines terechtkomen. Deze routines implementeren het default gedrag (en mogen enkel gebruik maken van de parameters, niet van de module variabelen);

- voorzie voor elke default routine een delegate- class waar deze toe behoort;

- definieer voor elke default routine een field van het overeenkomstige delegate type, en koppel dit field in de constructor aan zijn default implementatie;

- vervang elke oproep van de default-routines door een *invoke*- aanroep van het field. Dit start de methode in het field waarin default onze eigen routines te vinden zijn.

- voorzie *setters* voor elk delegate- field in de klasse. Dit geeft gebruikers van onze XMLViewer de kans om onze default methodes door zijn eigen routines te vervangen.

Figuur 4.13: XMLViewer en code plumbing

De linker versie toont het default gedrag van de XMLViewer. De rechter versie toont dezeldfe informatie in dezelfde uitvoering nadat de XMLViewer control werd voorzien van aangepaste visualisatie routines.

De screenshot toont een WPF- programma. De techniek op zich is uiteraard ook bruikbaar in een WinForms context.

lol, dat werkt!!!

4.7.2 Codeplumbing pitfalls

Wanneer we codeplumbing toepassen verdienen volgende punten uw aandacht:

- vermijd delegate properties indien u usercontrols maakt. Een property wordt door Visual Studio in het property window getoond (door de property te serialiseren). In het geval van een delegate zal dit in Visual Studio voor problemen zorgen (herhalende foutboodschappen, ook na het afsluiten en heropenen van het project in Visual Studio).

- laat uw codeplumbing routines *enkel* gebruik maken van de routine parameters. De delegate wordt immers uitgevoerd waar hij werd gedefinieerd. De module niveau variabelen zijn dus soms wel (wanneer de default routines worden uitgevoerd), soms niet (wanneer de nieuwe variant wordt uitgevoerd) deze van het object dat de delegate oproept.

4.8 Delegates: faits divers

4.8.1 Action en Func

Omdat in veel situaties gelijkaardige delegate definities nodig zijn werden in .NET een aantal delegate definities voorzien. Het is aan te raden deze indien mogelijk te gebruiken, eerder dan zelf nieuwe delegate klassen te definiëren;

4.8.1.1 Action

De klasse Action is een delegate klass voor routines zonder argumenten en zonder resultaat. Action$< T >$ is een (generische) klasse voor routines met exact één argument (gelijk welk type) en zonder resultaat.

4.8.1.2 Func

De generische klasse $Func< outTResult >$ wordt gebruikt voor functies zonder argumenten (en gelijk welk resultaat type), en $Func< inT, outTResult >$ wordt gebruikt voor functies met exact één argument (willekeurig type) en een willekeurig resultaat type.

lol, dat werkt!!!

4.8.2 Meer dan (begin)-invoke

Delegates ondersteunen meer dan enkel een (begin)invoke routine. Zo bestaat ook de mogelijkheid om verschillende delegates (van hetzelfde delegate type) te combineren (geen co- of contravariance mogelijk) en betaan er ook multicast delegates (die achter de schermen gebruikt worden om meerdere clients toe te laten op hetzelfde event te abonneren)

4.8.3 Timers

.NET ondersteunt verschillende Timer- classes (zie ook Comparing the Timer Classes in the .NET Framework Class Library). Indien u deze tekst doorneemt zal u begrijpen dat niet elke Timer 'tikt' op de UI-thread, wat betekent dat niet elke Timer geschikt is om UI-wijzigingen rechtstreeks aan te sturen (zie ook Asynchroon, met AsyncCallback, zonder UI).

Bemerk trouwens dat de constructor van de *System.Threading.Timer* klasse onder andere een delegate object verwacht. De twee andere timer-objecten werken met events, wat gezien Delegates en events eigenlijk niet echt een verschil uitmaakt.

4.8.4 Asynchrone eventverwerking

Wie de voorgaande topics goed begrepen heeft zou in staat moeten zijn om een eigen, asynchrone eventimplementatie op te zetten: een uitstekende oefening voor thuis.

4.8.5 Anonymous methods (.NET 2.0)

Bij het maken van een delegate instantie werd in voorgaande tekst telkens gebruikt gemaakt van een methode met gepaste signatuur. Indien men deze extra methode definitie wenst te vermijden (bijvoorbeeld bij het definiëren van MVVMCommands) kan men gebruik maken van *anonieme methodes*. Deze bestaan uit een (geschikte) formele parameterlijst definitie en een body:

```
btnAnonymousMethod.Click +=
    delegate(object o, RoutedEventArgs e)
        {MessageBox.Show("test1"); MessageBox.Show("test2");}
    ;
```

Figuur 4.14: Anonymous method voorbeeld

lol, dat werkt!!!

4.8.6 Lambda expressions (.NET 3.0)

Lambda expressions kunnen onder andere gebruikt worden om delegate objecten te creeëren. Ze bestaan uit twee stukken gescheiden door middel van de lambda operator =>. Links bevindt zich de input (het equivalent van de formele parameterlijst van een functiehoofding), rechts bevindt zich de body van de delegate. Het resultaat van de lambda expressie is (in dit geval) een delegate object:

```
btnAnonymousMethod.Click +=
    (object o, RoutedEventArgs e) =>
        {MessageBox.Show("test3"); MessageBox.Show("test4");
    ;
```

Figuur 4.15: Lambda expressie voorbeeld

Lambda expressions worden veel gebruikt in een LINQ-context, waar het injecteren van bijvoorbeeld selectielogica in een foreach-lus frequent voorkomt.

4.9 Delegates: niet voor elk programma een geschikte oplossing

- niet elk programma heeft nood aan delegates;
- (algemener:) niet elke 3nMCT-techniek is noodzakelijk om een goed programma te maken;
- (advies:) indien uw programma geen nood heeft aan delegates, gebruik ze dan niet: er zijn wellicht meer programma's te ontwikkelen zonder delegate- nood dan met.

4.10 Delegates: opgaves

- events: maak een eigen TextBox klasse met een extra event (guidelines!): dit event wordt afgevuurd indien het tekstvak leeg wordt. Demonstreer uw oplossing.
- multithreading: maak een window waarin verschillende tellers asynchroon kunnen oplopen. De teller wordt gerealiseerd door een control die erft van TextBox. Om dit te realiseren zal u de prioriteit van de visualisatie op background priority moeten uitvoeren. De demo toont de volgende elementen:
  - singlethreaded is er maar één activiteit actief;
  - multithreaded zijn er verschillende acties terzelfdertijd mogelijk: het tellen in twee textboxen en het verplaatsen van het window.
- maak een XMLViewer waarin de klant eigen routines kan inpluggen om specifieke xml's anders te visualiseren (zie ook XMLViewer). Demonstreer uw oplossing.

lol, dat werkt!!!

Jef Dacs
lol, dat werkt!!!

Hoofdstuk 5

Reflection

5.1 Reflection: duiding

Reflection is technologie waardoor (meta-) informatie omtrent programma-elementen programmatorisch toegankelijk wordt voor eigen programma's. Deze programma-elementen bevinden zich veelal in *assemblies* die niet tot het uitvoerende programma zelf behoren (externe modules). Op basis van de bekomen informatie zal ons eigen programma wellicht extra functionaliteit aanbieden. In dit hoofdstuk zullen we reflection gebruiken om een *plugin* infrastructuur op te zetten.

Hoewel reflection, net als delegates, de mogelijkheid biedt om routines te starten zou het verkeerd zijn om delegates als onderdeel van reflection te interpreteren (of omgekeerd). Een belangrijk verschil is mijns insziens de op voorhand gekende interface elementen van een delegate- instantie (*invoke*, ..) terwijl reflection net de mogelijkheid biedt om met at-compile-time ongekende klasses en/of routines te werken.

Het belang van reflection kan moeilijk overschat worden. Visual Studio illustreert mooi de mogelijkheden van reflection: intellisense, het property-window, de compiler .. : veel van de programmeur ondersteunende features maken gebruik van reflection, en in het bijzonder van attributen. Ook het hoofdstuk omtrent Aspect Oriented Programming steunt sterk op reflection.

Tot slot zal de in dit hoofdstuk opgedane kennis toelaten om object oriëntatie als een syntactische constructie bovenop procedureel programmeren te zien: een inzicht dat hopelijk de blik verruimt.

lol, dat werkt!!

5.1.1 Relevante namespaces

- *System.ComponentModel*
- *System.Reflection*

5.2 Reflection: attributen voorbeeld

Een programmeur kan code elementen voorzien van attributen om extra informatie aan het code element te koppelen. Deze attributen kunnen door andere programma's gelezen worden om een gepaste verwerking van het element mogelijk te maken. Deze verwerking staat los van de werking van het code element zelf die wellicht niet gewijzigd wordt. De namespace *System.ComponentModel* bevat attributen die door Visual Studio gebruikt worden. Een aantal van deze zijn enkel van toepassing in een WinForms context, en niet in WPF:

Figuur 5.1: ComponentModel attributen ($C^{\#}$)

Voor bovenstaande voorbeeld kunnen we volgende elementen vermelden:

- *DefaulEvent* en *DefaultProperty* worden enkel toegepast op klasse (usercontrol) niveau. Ze worden door Visual Studio gebruikt om in de designer volgende acties te ondernemen:
  - *DefaultProperty*: deze property wordt in het property window automatisch geslecteerd wanneer een control van deze klasse in de (Visual Studio-) designer wordt geselecteerd;
  - *DefaultEvent*: dit event wordt automatisch geselecteerd in het property window (event tab) wanneer een control van deze klasse in de (Visual Studio) designer wordt geselecteerd. Dubbel klikken op deze control zal code genereren om dit event op te vangen.
- *Category* is een attribuut waarmee de category (gebruikt in het propertywindow) van een property wordt opgegeven. Deze wordt door Visual Studio gebruikt (zie figuur);
- *Description* en *DefaultValue* zijn attributen die enkel in een Winforms Visual Studio designer omgeving gebruikt worden (de WPF designer in Visual Studio negeert deze attributen).

5.3 Reflection in .NET: meer dan attributen

5.3.1 Reflection in .NET: start

Maak een nieuwe solution met hierin een WPF-project. Maak een tweede(!) WPF project met volgende basis functionaliteit:

- voorzie een klasse *clsPersoon* met een aantal features;
- voorzie een knop op het default window om een nieuwe persoon aan te maken.

Compileer het nieuwe project en programmeer onderstaande code in het eerste project (zonder een referentie tussen beide projecten te leggen):

```
private void verwerkDemoUI()
    {
    Microsoft.Win32.OpenFileDialog ofd = new Microsoft.Win32.OpenFileDialog();
    ofd.Filter = ".NET assemblies|*.exe;*.dll";
    ofd.InitialDirectory = System.IO.Path.GetDirectoryName(
        System.Reflection.Assembly.GetExecutingAssembly().CodeBase);

    if (ofd.ShowDialog().Value)
        verwerkDemoUI(ofd.FileName);
    }
private void verwerkDemoUI(string sAssembly)
    {
    Assembly ass = Assembly.LoadFile(sAssembly);
    foreach (Type t in ass.GetTypes())
        Console.WriteLine(t.Name);
    }
```

Figuur 5.2: Assembly reflection ($C^{\#}$)

Indien we bij uitvoering van deze code en het tonen van de *OpenFileDialog* het compilatie resultaat van het tweede project selecteren, zal het uitvoerende programma in de console de naam van elk type in de gekozen assembly tonen: *MainWindow*, *App*, *Resources*, *Settings* en *clsPersoon*. Eenvoudig nazicht in Visual Studio leert dat dit klopt.

Bovenstaande code is een reflection startpunt: vertrekkend vanuit een assembly wordt informatie omtrent de code in de assembly opgevraagd (bijvoorbeeld de namen van de klasses). De Type-klasse ondersteunt flink wat getters: zo kunnen de attributen, constructoren, methodes, properties, events, ... worden opgevraagd. Elk van deze code elementen is terug ondervraagbaar. Het wordt sterk aangeraden om een aantal getters zelf te verwerken, ook al vind u hiervan geen afbeeldingen in deze tekst terug.

5.3.2 Hoe een instantie maken van een klasse in een assembly?

Normaal gezien zullen we, voor een gekende klasse clsX, volgende $C^{\#}$- syntax gebruiken:

```
clsX x = new clsX(..);
```

Figuur 5.3: Strongly typed instantiatie

In bovenstaande code wordt de gekende klasse clsX op twee plaatsen gebruikt: als type van de referentie en als type waarvan de constructor wordt opgeroepen. Indien we een type bekomen via reflection is bovenstaande onmogelijk: gezien het nodige type at compile time niet gekend is (clsX wordt gevonden in een at run-time geladen assembly) is bovenstaande lijn code onmogelijk (in $C^{\#}$), en moeten we twee problemen oplossen:

- het referentie type zullen we vervangen door *System.Object*. Omdat elke klasse (al dan niet indirect) erft van *System.Object* is elk instantie van een klasse te koppelen aan een referentie van dit root- type (polymorfisme door overerving). Natuurlijk kunnen we, indien we extra kennis omtrent de klasses hebben, eventueel een meer gespecialiseerd type gebruiken (zie het voorbeeld hieronder);
- het oproepen van de constructor van een klasse zal gecodeerd worden via de *Activator*-klasse:

```
private void verwerkIndienWindow(Type t)
    {
    if (t.Name.Equals("MainWindow"))    //dergelijke test vermijden => attributes
        {
        Window wdw = (Window) Activator.CreateInstance(t);
        wdw.Show();
        }
    }
```

Figuur 5.4: Activator.CreateInstance ($C^{\#}$)

- indien de constructor argumenten vereist kunnen deze aan de *CreateInstance*- oproep worden meegegeven (u bent *niet* beperkt tot parameterloze constructoren);
- omdat we weten dat een klasse met deze naam (beslissingen op basis van een klasse naam zijn te vermijden!) een Window is gebruiken we een *Window* referentie waarvan de *Show*-method *at compile time* gekend is.

Jeroen Werkt!!!

Een iets gecompliceerder voorbeeld illustreert het starten van een methode vertrekkend vanuit zijn object referentie:

```
private void verwerkIndienPersoon(Type t)
    {
    if (t.Name.Equals("clsPersoon"))  //dergelijke test vermijden => attributen
        {
        object p = Activator.CreateInstance(t);
        t.InvokeMember("FNaam", BindingFlags.SetProperty, null, p, new object[] { "Familie" });
        t.InvokeMember("VNaam", BindingFlags.SetProperty, null, p, new object[] { "lid" });

        string sPersoon = (string) t.InvokeMember("ToString", BindingFlags.InvokeMethod, null, p, new object[] { });
        Console.WriteLine(sPersoon);
        }
```

Figuur 5.5: Activator.InvokeMember ($C^{\#}$)

- de Type- klasse ondersteunt een *InvokeMember* methode waarmee een methode kan gestart worden. Het object dat de methode uitvoert is een argument van deze oproep, de nodige argumenten worden als een array van objecten voorzien;
- deze manier van werken is case sensitief!
- deze voorbeeld code is weinig representatief: de kans is klein dat de klasse- naam op voorhand gekend is, net zomin als men mag verwachten de methodes op voorhand te kennen. Indien de methodes op voorhand gekend zijn is het wellicht aangewezen om met een interface te werken. Op die manier is een verwerking als deze van het window (zie vorige voorbeeld) mogelijk.

lol, dat werkt!!!

5.3.3 .NET en attributes

5.3.3.1 .NET en attribute informatie opvragen

De klasse *Type* ondersteunt een *GetCustomAttributes* methode waarmee de (custom-) attributen van een klasse kunnen opgevraagd worden. Ook andere code elementen hebben een *GetCustomAttributes* property. Op basis van de extra informatie gevonden in een attribuut kan een gebruikers programma gestuurd worden. Verschillende attribute klasses ondersteunen natuurlijk verschillende properties. Een voorbeeld attribuut zullen we uitwerken in onze eigen plugin infrastructuur implementatie.

5.3.3.2 Hoe zelf Attribute klasses maken in .NET?

Een goede beschrijving kan u vinden op Writing Custom Attributes. Als opmerking hierbij zou ik formuleren dat ik verwacht dat de meeste properties *read-only* gedefinieerd zijn (en niet *read-write* zoals in de voorbeeldcode).

Er wordt zeker verwacht dat u in staat bent om attributen te definiëren die enkel op classes of enkel op methodes toegepast kunnen worden.

5.4 Een eigen plugin- infrastructuur!

5.4.1 Een eigen plugin- infrastructuur: vereisten

Ik wens volgende situatie te realiseren:

- we moeten de mogelijkheid voorzien om in een toepassing extra logica uit een op voorhand niet gekende *assembly* te laden;
- de extra logica wordt aangeboden door middel van plugins. Een plugin- klasse erft van *Window*, en alle relevante extra's worden ontsloten via deze klasses. Er kunnen Window klasses in de assembly worden gevonden die geen plugins zijn (maar bijvoorbeeld gestart kunnen worden door een plugin in dezelfde assembly);
- het is de bedoeling om bij het laden van een assembly alle gevonden plugins via het menu te presenteren. De menu-tekst voor een plugin moet ook in de assembly gevonden worden;
- zie demo.

Ik, dat werkt!!

5.4.2 Een eigen plugin- infrastructuur: mogelijke oplossing

Voorgaande vereisten kunnen op verschillende wijzes gerealiseerd worden. De hier geformuleerde oplossing is attribute- gebaseerd:

- het onderscheid maken (differentiëren) tussen klasses die al dan niet als plugin moeten aanzien worden kan door middel van een attribuut. Dit attribuut ondersteunt twee properties: het al dan niet plugin zijn (isPlugin, bool), en de omschrijving van de eventuele plugin (description, String). Dit resulteert in een plugin- klasse met twee fields, twee (read only) properties en een constructor met twee argumenten;
  - indien een klasse dit attribuut niet heeft is de klasse geen plugin (default).
- indien een klasse het plugin attribuut heeft, en isPlugin is true, dan is dit een plugin klasse waarvoor we een extra menu item zullen voorzien (in de software die de plugin-assembly verwerkt). Indien op dit menu item wordt geklikt wordt een instantie van de plugin klasse aangemaakt en getoond
  - we veronderstellen (leggen op) dat het plugin attribuut enkel is toegepast geworden op Window- klasses;
  - het is *niet* de bedoeling om in het client- window een boekhouding tussen menuitems en plugin- klasses bij te houden: het menu-item zelf moet bij het klikken alle nodige bewerkingen onafhankelijk kunnen afhandelen.
    - dit laatste punt is belangrijk en lijkt soms in de totaal oefening verloren te gaan.

5.4.3 Een eigen plugin- infrastructuur: implementatie

Zowel de plugin- assembly als het client- programma (het programma dat de plugins wil laden) moeten toegang hebben tot dezelfde plugin- attribuut klasse. Omdat er geen referentie tussen de plugin- en de client- assembly mag liggen (anders is de plugin assembly at compile time gekend en is de ganse plugin- opzet overbodig) en beide assemblies toch dezelfde attribuutklasse moeten kennen zijn we verplicht het plugin- attribuut in een apart project te plaatsen (waar zowel het plugin- project als het client- project naar refereren). We hebben dus minimum drie projecten! Omtrent die drie projecten kunnen we volgende vragen stellen:

- welk project type is elk project?
- welke referenties liggen tussen deze projecten?
- waarom hebben we eigenlijk drie projecten nodig? Indien het plugin- en client project beide dezelfde bron-file voor de attribuut klasse bevatten, is dit niet voldoende?
- formuleer een oplossing met maar twee projecten (perfect mogelijk trouwens, minder goed, naam gebaseerd).

Maak in dit derde project een attribuut klasse met volgende eigenschappen:

- volg de naamgeving guideline!
- voorzie de mogelijkheid om het plugin zijn en een plugin- omschrijving te bewaren, toe te kennen via de constructor en op te vragen via read only properties;
- maak het onmogelijk om dit attribuut toe te passen op andere code elementen dan klasse definities.

Maak in het tweede project (het plugin- project dat plugins aanlevert) een window aan en koppel wat logica aan het klikken op een knop. Plaats het plugin- attribuut op de MainWindow klasse en niet op de klasse clsPersoon.

Het clientproject dat de eventuele plugins zal laden gaat als volgt tewerk:

- bij het opstarten van een window dat de plugins ter beschikking stelt (de constructor van dit window is een goede plaats) lezen we alle assembly files gevonden op een vooraf afgesproken plaats (in mijn eigen oplossing is dit de plugins subdirectory van de toepassing)
  - System.IO is uw favoriete namespace voor dergelijke zaken;
  - gebruik een filter om enkel de assembly files op te vragen;
  - in een echte toepassing zou de plugin-directory configureerbaar moeten zijn.
- de types in elke *dll*- assembly worden overlopen. Indien ze voorzien zijn van een plugin attribuut (en isPlugin is *true*) voorzien we een extra menu item in het (main-)menu van dit window. Dit nieuwe menu-item is een instantie van een eigen menuitem klasse:

lol, dat werkt!!!

  - ze erft van de klasse *MenuItem*;
  - de omschrijving getoond in het menu is de description gevonden in het plugin attribuut;
  - ze implementeert een eventhandler zodat bij het klikken op een menu-item van deze klasse een istantie van het geassocieerde plugin-type wordt gemaakt en getoond;
  - bepaal zelf de nodige constructor argumenten.

5.4.4 Eigen plugin infrastructuur: demo

Gedemonstreerde elementen:

- het verwijderen van plugin assemblies verwijdert de plugins uit het menu;
- indien *isPlugin false* terug geeft wordt de eventuele plugin niet getoond;
- het verwijderen van het plugin attribuut zal (na hercompilatie) de plugin niet meer tonen in het menu;
- het plaatsen van het plugin- attribuut op een niet window klasse resulteert uiteraard in een foute werking. Een goed programma zal hierop niet crashen!
- er wordt in het window met de menu-items geen koppeling tussen de menu items en het plugin type voorzien: de volledige afhandeling om een plugin window te starten dient in de eigen plugin menu item-klasse geprogrammeerd te zijn.

lol, dat werkt!!!

5.5 Namespaces met interessante attributen

- System.Diagnostics: vooral debug en performance attributes;
  - ConditionalAttribute: zeer interessant wanneer u in uw code het onderscheid wensts te maken tussen Debug- en Release- uitvoeringen (klik nu op de link :)).
- System.Componentmodel: vooral component en control gerichte klasses (zie msdn);
  - onder andere tal van convertoren (niet echt IDE stuff);
  - licensing attributen (!);

5.6 Reflection: opgaves

- ontwerp een plugin infrastructuur (analoog met de demo);
- pas het fractalen programma aan zodat het tonen van de drietallige boom en Sierpinski als plugin in een programma wordt opgenomen.

Hoofdstuk 6

Runtime compilatie

6.1 Runtime compilatie: duiding

De meeste softwareontwikkelingen bestaan uit twee stappen: er wordt code geschreven (of gegenereerd) (stap één), waarna deze gecompileerd wordt tot een uitvoerbaar programma (stap twee). Eventueel zal een toepassing verschillende projecten bevatten die elk gecodeerd en gecompileerd worden. Dit beperkt uiteraard de mogelijkheden van de toepassing: wat niet op voorhand werd geprogrammeerd (eventueel in een PlugIn) kan niet uitgevoerd worden. Indien we een tekenprogramma wensen te ontwikkelen, waarbij de gebruiker het functievoorschrift intikt hebben we een probleem (we wensen niet zelf een wiskunde parser te schrijven). Ook indien we een toepassing maken waarbij de beveiliging van sommige onderdelen door de klant wordt bepaald kunnen we niet altijd alle mogelijkheden op voorhand voorzien. Sommige database toepassingen zullen een property list in een tabel bewaren (bv.: de talen waarin een produkt gekend moet zijn, zie ook de vertalingen demo): ook hier is het niet mogelijk om op voorhand alle properties in een klasse te compileren. In dergelijke situaties is het interessant om, gedurende de uitvoering van het programma, een extra stukje code te genereren, compileren en uitvoeren: *runtime compilatie* (ook wel on the fly compilatie geheten), zodat databinding zijn rol kan spelen.

lol, dat werkt!!

Recentelijk werden de .NET compilers herschreven (Roslyn) zodat u indien u dat wenst de compiler informatie kan bekomen en wijzigen. Hoewel dit allerhande code manipulerende mogelijkheden biedt heb ik er nog geen gevonden waar ik het wenselijk acht om een eigen implementatie uit te werken.

6.1.1 Relevante namespaces

- *System.CodeDom.Compiler*

6.2 .NET compilaties starten

Zowel de $C^{\#}$als de VB.NET compilers maken deel uit van de CLR. Afhankelijk van de gegenereerde code ($C^{\#}$of VB.NET) zullen we dan ook de juiste compiler moeten oproepen. We zullen de code en het compilatieresultaat (de assembly) in memory houden zodat er geen files overblijven nadat ons programma eindigt.

6.2.1 Command line compilatie

Wellicht compileerde u tot nu toe altijd met behulp van Visual Studio. Het achterliggende compilatie proces (in welke volgorde compileren, welke compilatie parameters zijn nodig) wordt volledig verborgen (een voordeel). Om het compilatie proces duidelijker te maken zullen we in een eerste stap een volledig (console-) programma editeren, compileren en uitvoeren zonder gebruik te maken van Visual Studio.

6.2.1.1 Command line compilatie: $C^{\#}$

- maak een testfile abc.cs met een ascii- tekstverwerker (uw code files mogen enkel programma code bevatten, een tekstverwerker als Word is dus niet geschikt) en codeer volgende elementen ($C^{\#}$is case sensitive!):
  - voorzie een gepaste Main- methode (zie ook programmeren II: elk programma start door het uitvoeren van een static Main methode; of lees Main() and Command Line Arguments ($C^{\#}$Programming Guide)). Uiteraard moet u deze methode in een klasse coderen. Anders dan in JAVA moet de klasse naam niet identiek zijn aan de filenaam;
    - de klasse *String* zal enkel gekend zijn indien u ze prefixt met de namespace *System* of wanneer u een overeenkomstig *using* statement codeert;
  - schrijf het eerste runtime argument weg in de Console;

```
using System;
public class Test
        {
        public static void Main(String[] args)
                {
                Console.WriteLine(args[0]);
                }
        }
```

Figuur 6.1: Commandline compilatie: $C^{\#}$broncode

lol, dat werkt!!!

- start een Visual Studio command prompt (
  - win8: tik op het startscherm *tools* en kies de *native tools command prompt*;
  - win7: Start.All Programs.Microsoft Visual Studio xxx.Visual Studio Tools.Visual Studio Command prompt(xxx);
  - csc.exe is de $C^{\#}$compiler. U dient deze op te roepen met de juiste set parameters (*csc /?* toont u alle mogelijke opties met een summiere toelichting). Voor ons eenvoudige programma is de */out:* parameter eigenlijk overbodig (indien geen out- parameter wordt opgegeven heeft de resulterende assembly dezelfde naam als de bron-file);
  - met de target- parameter (niet in het voorbeeld) kan u aangeven of u een .exe (default) of .dll assembly wenst te bekomen;

Figuur 6.2: $C^{\#}$commandline compilatie en uitvoering

- indien ons programma gebruik zou maken van niet standaard bibliotheken moeten deze bij compilatie uiteraard ook worden meegegeven. Voor een gewoon WinForms of WPF programma wordt de parameterlijst zo groot (ongeveer elke project property komt terug als compiler optie), en is de compilatie volgorde zo belangrijk dat we dit zeker niet manueel wensen uit te voeren. Gedurende het programmeren zal Visual Studio dit voor ons doen. Indien u het proces wenst te automatiseren (om een groot project elke nacht volledig te hercompileren) kan u zich best wat verdiepen in MSBuild.

6.2.1.2 Command line compilatie: VB.NET

We volgen gelijkaardige stappen, maar dan voor VB.NET:

- VB.NET code:

```
imports System

public class test
   public shared sub main(args as String())
        Console.writeline(args(0))
   end sub
end class
```

Figuur 6.3: Commandline compilatie: VB.NET broncode

- VB.NET compilatie en uitvoering:

Figuur 6.4: VB.NET commandline compilatie voorbeeld

Het coderen en compileren is voor beide programmeertalen sterk gelijkaardig.

6.2.1.3 Een extraatje omtrent MSBuild

Het compileer proces van MSBuild gebeurt door middel van informatie gevonden in tekst files (zie ook MSBuild .Targets Files). Indien u hierin onderlegt bent kan u deze aanpassen (neen, u bent het wellicht niet). Aanpassen van deze files is de manier waarop PostSharp er in slaagt om het compilatie proces aan te passen zonder dat u als programmeur dient tussen te komen.

Jef Lewis
lol, dat werkt!!!

6.2.2 Runtime compilatie: System.CodeDom.Compiler

Het is natuurlijk niet de bedoeling om voor normale sitaties een eigen compilatie proces uit te werken. Het is wel onze bedoeling om, vanuit een programma, programmacode te genereren en compileren (tot een .dll of .exe), en de bekomen assembly te verwerken zoals reeds werd gezien in reflection.

De .NET compilers zijn beschikbaar via de *System.CodeDom*- namespace:

System.CodeDom.Compiler Namespace

The **System.CodeDom.Compiler** namespace contains types for managing the generation and compilation of source code in supported programming languages. Code generators can each produce source code in a particular programming language based on the structure of Code Document Object Model (CodeDOM) source code models consisting of elements provided by the System.CodeDom namespace.

Classes

Class	Description
CodeCompiler	Provides an example implementation of the ICodeCompiler interface.
CodeDomProvider	Provides a base class for CodeDomProvider implementations. This class is abstract.
CodeGenerator	Provides an example implementation of the ICodeGenerator interface. This class is abstract.
CodeGeneratorOptions	Represents a set of options used by a code generator.
CodeParser	Provides an empty implementation of the ICodeParser interface.
CompilerError	Represents a compiler error or warning.
CompilerErrorCollection	Represents a collection of CompilerError objects.
CompilerInfo	Represents the configuration settings of a language provider. This class cannot be inherited.
CompilerParameters	Represents the parameters used to invoke a compiler.
CompilerResults	Represents the results of compilation that are returned from a compiler.
Executor	Provides command execution functions for invoking compilers. This class cannot be inherited.
GeneratedCodeAttribute	Identifies code generated by a tool. This class cannot be inherited.
IndentedTextWriter	Provides a text writer that can indent new lines by a tab string token.
TempFileCollection	Represents a collection of temporary files.

Figuur 6.5: *C#*System.CodeDom.Compiler namespace

Bemerk dat er geen aparte klasses per programmeertaal zijn:

- de klasse *CodeDomProvider* geeft toegang tot de .NET- compilers. De static methode *CreateProvider* heeft een argument waarmee wordt aangegeven voor welke programmeertaal we een compiler wensen te bekomen;
  - indien u twijfelt omtrent de taal-parametertekst (c# of csharp? VB of VB.NET? case sensitive?) kan u deze bekomen door de static methode *GetLanguageFromExtension* op te roepen (de file extentie moet u dan uiteraard wel kennen);
- *CompilerParameters* is handiger dan de commandline benadering: de verschillende parameters zijn niet als een lange tekst te formuleren maar zijn properties op een CompilerParameters object;
  - het toevoegen van een assembly kan gebeuren door middel van zijn naam: *System.Data.dll*
  - een assembly waarin een type XYZ zich bevind kan worden teruggevonden door *typeof(XYZ).Assembly.Location*

- *CompilerResults* bevat het compilatie resultaat. Dit bevat zowel de compilatie feedback (error, warning, ..) als de (eventueel) resulterende assembly;

6.3 CodeDom

De eerder aan bod gekomen oplossing verwacht broncode die gecompileerd kan worden: deze is natuurlijk programmeertaal afhankelijk. In een andere benadering kunnen we een programmeer boom maken waarin programmeer- entiteiten worden gehangen (types, methodes, variabelen, commando's, ..) waarvoor achteraf programma code (in de programmeertaal naar keuze) wordt gegenereerd:

```
CodeCompileUnit ccunit = new CodeCompileUnit();
CodeNamespace cns = new CodeNamespace("taalprog");
ccunit.Namespaces.Add(cns);

CodeTypeDeclaration klas = new CodeTypeDeclaration("klasnaam");
cns.Types.Add(klas);
klas.Attributes = MemberAttributes.Public;
```

Figuur 6.6: CodeDOM

We spreken in deze context over een abstract syntax tree.

Jef Baels
lol, dat werk

6.4 System.Reflection.emit

Voor de volledigheid vermelden we ook de System.Reflection.Emit Namespace. Deze laat toe om bij uitvoering *IL*- code genereren (normaal gezien het resultaat van een compilatie). Dit valt jammer genoeg buiten het bereik van deze cursus en mijn kennis.

6.5 Het Roslyn project

Een interessante ontwikkeling in het compilatie gebeuren van .NET is de ontwikkeling van het Roslyn project (momenteel nog in CTP (community technical preview)). Hierbij worden tussentijdse resultaten (de abstract syntax tree, koppeling tussen namen en objecten, ..) van het compilatie proces ontsloten naar ontwikkelaars toe. Dit biedt gigantisch veel mogelijkheden naar geautomatiseerde code processing (zeker naar rapportering toe, wellicht ook naar het automatisch aanpassen van code toe).

6.6 Assembly code inspecteren

Het resultaat van een .NET compilatie is (indien gelukt) een assembly. Deze bestaat uit *IL*(Intermediate language) code: de machinetaal van .NET. Deze assemblies kunnen door middel van disassemblers gemakkelijk gedecompileerd worden. Dit laat derden toe om uw broncode grotendeels te reproduceren wat meestal niet wenselijk is. Het gebruik van obfuscators kan de leesbaarheid van de gedecompileerde broncode sterk omlaag halen.

Deze problematiek is niet .NET specifiek: ook byte code (het java equivalent van IL- code) kan gedecompileerd worden. Ook voor C(++) output zijn er disassemblers te vinden. In de .NET context zal u merken dat u een assembly kan decompileren naar uw programmeertaal naar keus, wat ook de oorspronkelijke programmeertaal was.

6.6.1 ILDasm

ILDasm is de *IL* disassembler meegeleverd met *Visual Studio*. Deze kan gebruikt worden om *IL*- code te tonen in een textfile formaat. Op deze manier is het mogelijk om bijvoorbeeld de aanwezigheid van attributen na te kijken. Hoewel *ILDasm* zeker zijn plaats heeft (zie ook Ildasm.exe (MSIL Disassembler)) is het zeker niet de meest geschikte tool om de brontekst van een programma na te kijken:

Figuur 6.7: ILDasm output

6.6.2 ILSpy

ILSpy is het gratis antwoord waarmee werd gestart nadat de voorheen gratis Reflector software betalend werd. Voor zover ik beide oppervlakkig gebruikt heb vind ik ze sterk gelijkwaardig: beide laten toe om een door ons gecompileerd programma om te vormen naar een .NET-taal naar keuze.

Bij het downloaden van ILSpy kan u kiezen tussen de broncode en de gecompileerde bestanden. Ikzelf verkies meestal de broncode, maar het staat u uiteraard vrij om de binaries te installeren.

Gelieve na het installeren *ILSpy* eens op te starten en een programma te decompileren:

Figuur 6.8: ILSpy output

Bemerk dat in bovenstaande figuur VB.NET- code werd gegenereerd terwijl de oorspronkelijke programmatekst in $C^{\#}$werd geschreven.

6.6.3 Reflector

Deze (niet langer gratis) tool is (niet langer) wat u zoekt! Hiermee bent u in staat om een assembly te decompileren in *de programmeertaal naar keus*. Indien u de nodige plugins installeert kan u een assembly volledig decompileren tot een project (sommige namen zullen wel niet compileerbaar zijn, maar dat is voor kwaadwillige derden slechts een kleine beperking):

Figuur 6.9: Reflector output

Bemerk dat in bovenstaande figuur VB.NET- code werd gegenereerd terwijl de oorspronkelijke programmatekst in $C^{\#}$werd geschreven. In de programma code zijn de groene tekst elementen hyperlinks (niet beschikbaar in ILSpy): hiermee kan u navigeren naar de definitie van dit element (ongeveer zoals *F12* in Visual Studio).

6.6.4 Obfuscatie

Obfuscatie is een techniek om een assembly aan te maken die ook na decompilatie quasi onleesbaar is. De (gratis) versie die geïnstalleerd wordt met Visual Studio (onder tools) vertrekt van een compilatie resultaat en zal alle hierin gevonden namen (klasses, methodes, parameters, ..) *scramblen*, wat het geheel voor een complex programma quasi onleesbaar maakt. Het is uiteraard niet wenselijk om de broncode van het project zelf te scramblen, omdat u in dat geval geen aanpassingen meer zou kunnen doorvoeren.

Figuur 6.10: Obfuscated output via Reflector

Bemerk in bovenstaande figuur het hernoemd zijn van de controls. Veel routine namen zijn niet hernoemd geworden, wat spijtig is. Wellicht zal de betalende versie hier een betere obfuscatie mogelijk maken.

6.6.4.1 Confuser

Een betere obfuscater vindt u op Confuser. Het resultaat van deze obfuscater kan niet geopend worden in ILDASM en is totaal onleesbaar wanneer u bijvoorbeeld ILSpy gebruikt.

6.7 Opgaves

6.7.1 Een tekenprogramma

Het te realiseren programma moet aan volgende vereisten voldoen:

- de gebruiker kan zelf een functievoorschrift intikken (reden waarom we on-the-fly compilatie nodig hebben: we zullen dit functievoorschrift compileren zodat we het kunnen uitvoeren);
- het tekenen gebeurt in een usercontrol, waarbinnen de tekening geschaald wordt getoond: we rekken (of krimpen) de te tekenen figuur zodat de bovenkant van de figuur de bovenkant van de usercontrol raakt (idem voor de onderkant);

Figuur 6.11: Tekenen van een functie

lol, dat ...!!

6.7.2 ILSpy

Op het examen wordt verwacht dat u volgende manipulaties vlot kan uitvoeren:

- installeer ILSpy en pas deze toe op een assembly naar keus;
- demonstreer dat u kan decompileren naar verschillende programmeer talen.

6.7.3 Obfuscatie

Op het examen wordt verwacht dat u volgende manipulaties vlot kan uitvoeren:

- gebruik *Confuser* om een assembly naar keus te obfuscaten;
- gebruik de obfuscator (gevonden onder tools) om een assembly naar keus te obfuscaten;
- gebruik ILSpy om het obfuscated zijn van uw nieuwe assembly te demonstreren.

Jef Daels
lol, dat werkt!!!

Hoofdstuk 7

Backtracking

7.1 Backtracking: duiding

De eerste computers werden oorspronkelijk gebruikt om oplossingen te bieden voor problemen die mensen niet (snel genoeg) konden oplossen. Dit hoofdstuk behandelt het gebruik van een computer om problemen op te lossen waarvoor we zelf, zonder computer, geen antwoord kunnen formuleren. De gebruikte techniek heet *backtracking*: hij is niet op alle problemen toepasbaar, is soms hopeloos inefficient maar is een haalbare kaart om artificiële intelligentie te introduceren. In de laatste sectie zullen we backtracking gebruiken om AI in een spelletjes context te introduceren.

lol, dat werkt!!

7.1.1 Relevante namespaces

7.1.2 Backtracking: principe

Backtracking is een benadering waarbij men, door het overlopen van alle mogelijke oplossingen, zoekt naar een beste (eerste, ..) oplossing. De implementatie van het algorimte is meestal recursief, met hierin volgende stappen:

- *stopconditie*: indien de oplossing werd gevonden geven we true terug;
- anders worden alle mogelijkheden voor deze stap in een lus overlopen:
  - voer één van de stap mogelijkheden uit;
  - indien de recursieve oproep voor de volgende stap slaagt hebben we een oplossing gevonden: we geven true terug;
  - indien de recursieve oproep faalt maken we de zopas gezette stap ongedaan en vervangen deze door één van de andere mogelijkheden, en herhalen de recursieve stap;
- indien geen enkele van de mogelijkheden tot een oplossing leidt moeten we concluderen dat de eerder gezette stappen het vinden van een oplossing blokkeren, en keren we terug naar de oproeper (return false), die dan op zijn beurt een volgende mogelijkheid kan onderzoeken.

7.2 N-Queens probleem

7.2.1 Probleemstelling, synchrone oplossing

Een gemakkelijk uit te leggen backtrack probleem is het N-Queens probleem: hoe plaats ik N koninginnen op een schaakbord zodat deze elkaar niet slaan. Twee koninginnen slaan elkaar indien ze:

- op dezelfde kolom staan;
- op dezelfde rij staan;
- uit de eerste koningin een diagonaal vertrekt waarop de tweede koningin zich bevindt.

Een geïllustreerde probleemstelling vind u op Eight queens puzzle. De toelichting van het algoritme gebeurt tijdens de les.

Figuur 7.1: NQueens oplossen

7.2.2 Asynchroon: BeginInvoke/4

Indien we grotere aantallen (bijvoorbeeld 30) koninginnen dienen te plaatsen neemt de berekening flink wat tijd in beslag. Om de interface hiermee niet te blokkeren zullen we de berekening van de oplossing op een andere thread laten uitvoeren. Hiertoe gaan we als volgt te werk:

- maak een delegate-class voor de zoekroutine;
- start een asynchrone oproep voor deze zoekroutine, en voorzie een callback waarmee het berekende resultaat in de UI kan getoond worden (u zal hoogst waarschijnlijk een state object nodig hebben);
- laat de asynchrone thread de berekende oplossing in de UI tonen (let op de UI-thread pitfall!).

Hint: om te vermijden dat gedurende de asynchrone zoekoperatie een volgende zoekoperatie gestart zou worden (er kan immers geklikt worden in de UI terwijl de zoekoperatie loopt) is het best om de UI-elementen die de zoekopdracht starten of specifiëren disabled worden geplaatst: het is immers niet wenselijk de zoek- informatie te wijzigen terwijl de vorige zoekopdracht nog loopt (de getoonde resultaten zouden niet langer overeenstemmen met de zichtbare criteria) net zomin als we een volgende zoekopdracht willen starten terwijl de vorige nog niet ten einde was. Als oefening kan eventueel een cancel- button voorzien worden.

7.2.3 Asynchroon: async/await

Hiervoor verwijs ik naar Await/Async verder in deze tekst. Deze is ook bruikbaar voor de Sudoku- toepassing die straks aan bod komt.

7.3 Sudoku's oplossen

lol, dat werkt!!!

Het oplossen van Sudoku's is niet altijd eenvoudig. Een computerprogramma kan u behulpzaam zijn, maar ontneemt u wellicht het liefhebbers plezier.

7.3.1 Sudoku: spelregels

Het invullen van een 9x9 sudoku gebeurt op volgende wijze:

- elk hokje van de matrix moet worden ingevuld met een geheel getal tussen 1 en 9 (grenzen inbegrepen);
- in een rij mogen geen dubbels voorkomen;
- in een kolom mogen geen dubbels voorkomen;
- in elk van de negen 3x3 hokjes (naast elkaar, niet overlappend) mogen er geen dubbels voorkomen;
- een opgave toont een 9x9 rooster met een aantal ingevulde velden:

		5	2	3	6		8	9
						2		
2								
3			4					
			8	9		3	2	7
	9		3			4	5	
		1	6	8				
					5			
		3	7					

Figuur 7.2: Sudoku spelregels: een opgave

- de oplossing is gevonden wanneer de speler het rooster correct aanvult:

1	4	5	2	3	6	7	8	9
6	3	7	9	1	8	2	4	5
2	8	9	5	7	4	1	3	6
3	1	2	4	5	7	6	9	8
4	5	6	8	9	1	3	2	7
7	9	8	3	6	2	4	5	1
5	2	1	6	8	3	9	7	4
9	7	4	1	2	5	8	6	3
8	6	3	7	4	9	5	1	2

Figuur 7.3: Sudoku spelregels: een oplossing

7.3.2 Software vereisten

- UI vereisten:
  - enkel numerieke input, beperkt tot één positie is mogelijk. Het is niet de bedoeling feedback te geven indien de gebuiker hiervan afwijkt, het is de bedoeling dat hij er niet kan van afwijken (zie ook TXTINT;
  - er is een intuitieve navigatie op basis van de pijltjes mogelijk;
  - er is kleurverschil tussen opgave en oplossing;
  - bij ingave van een foutief getal wordt dit (discreet) gekleurd.
  - de gebruiker kan de opgave niet wijzigen gedurende het oplossen
- Sudoku vereisten:
  - we kunnen opgaves ingeven en bewaren (op file), en deze later terug inlezen;
  - een opgave kan door het programma correct worden opgelost (indien er een oplossing bestaat).

lol, dat werkt!!!
Joe Raels

7.3.3 Sudoku oplos algoritme

De recursieve oproep *zoekOplossing(Cell cell,int[,]bord)* kent twee parameters:

- *cell* is de cell die we nu wensen in te vullen. Deze stuurt de stopconditie: indien de cel die we wensen in te vullen buiten het bord valt zijn alle bord cellen ingevuld en hebben we een oplossing;
  - *Cell* is een eigen klasse (we houden rij- en kolom- waardes bij (Integers). De klasse Point werd niet gebruikt omdat deze doubles bevat);
- *bord* bevat de bordsituatie zoals ze nu gekend is. Deze situatie hebben we nodig om het correct zijn van een nieuwe zet te beoordelen. Omdat in het bord de oplossing die we zoeken verder wordt opgebouwd bij elke recursieve stap spreken we van een *accumulerende parameter*.

- het return type is *boolean*: *true* indien een oplossing wordt gevonden, *false* anders.

Het algoritme gaat als volgt:

- indien *cell* buiten het bord valt zijn alle velden ingevuld en hebben we een oplossing in *bord*;
- overloop elke mogelijke waarde voor cell en voer volgende stappen uit:
  - vul de waarde in op de nieuwe positie;
    - indien dit een Sudoku probleem veroorzaakt gaan we over naar de volgende waarde;
    - indien dit geen probleem veroorzaakt voeren we een recursieve oproep uit voor de volgende cel. Indien deze oproep *true* teruggeeft vonden we een oplossing (in bord) en returnen we zelf ook *true*. In het andere geval (de recursieve oproep levert false) was het onmogelijk om de rest van het bord in te vullen op een correcte manier;
  - verwijder de zopas gezette waarde en probeer een volgende mogelijkheid;
- indien na het overlopen van alle mogelijkheden nog geen oplossing werd gevonden moeten we false terug geven: het was niet mogelijk de Sudoku verder op te lossen: we returnen false waardoor de oproeper zijn laatste poging ongedaan maakt en een andere poging zal ondernemen.

Heb speciaal aandacht voor volgende elementen:

- een zet die niet voldoet moet terug van het bord genomen worden;
- indien een oplossing wordt gevonden mag de bord parameter (die deze oplossing bevat) niet meer gewijzigd worden.

Dit algoritme wordt ook aan bord op basis van voorbeelden ontwikkeld. Ik heb niet de ambitie backtracking volledig schriftelijke toe te lichten.

lol, dat werkt!!!

7.3.3.1 Constraint satisfaction solver

Voorgaande is niet de enige manier om Sudoku's op te lossen. Een interessante alternatieve benadering wordt beschreven in Sudoku using Microsoft solver foundation. Het onderliggende algoritme zou opnieuw backtracking kunnen zijn.

7.4 Sudoku routed events introductie

Het is de bedoeling om pijltjes navigatie op het Sudoku bord toe te laten: we vangen het *PreviewKeyDownEvent* op en voeren de eventuele gepaste navigatie uit. Elke TextBox op het Sudoku-bord (er zijn er 81) kan de bron van dergelijk event zijn: in een klassieke benadering moeten minstens 81 eventhandlers gekoppeld worden. Indien men beslist andere controls dan TextBoxen te gebruiken moet men events van deze andere controls opvangen. Om in dergelijke situaties een elegante oplossing te bieden werden *routed events* geïntroduceerd:

- een WPF-control kan zich abonneren op een *routed event* gedefinieerd op een andere klasse (zoals een DependencyObject een DependencyProperty gedefinieerd op een andere klasse kan hebben):

```
public ccSudoku9x9()
    {
    this.Background = new SolidColorBrush(Colors.Transparent);
    initSudoku9x9();
    zetContextMenu();

    this.AddHandler(TextBox.TextChangedEvent, new TextChangedEventHandler( text_changed));
    this.AddHandler(TextBox.PreviewKeyDownEvent, new KeyEventHandler(keyDown));
    }
```

Figuur 7.4: Sudoku routed events demo

- de visual tree wordt eerst van container naar child elementen doorlopen (de *preview*-events): de *tunneling* fase;
- daarna wordt de visual tree van child naar container element doorlopen (de gewone event namen): de *bubbling* fase;
- indien een eventhandler (zowel tunneling als bubbling) het event afhandelt kan de verdere eventhandling afgebroken worden (handled=true op het argument e);
  - dit afbreken is op zijn beurt terug te omzeilen;
  - door het toepassen van templates en styles kan de visual tree behoorlijk complex worden. Toch wordt hij volledig doorlopen.
- hoewel *ccSudoku9x9* noch een *TextChangedEvent* noch een *PreviewKeyDownEvent* kent kan hij toch subscriben op deze routed events. In zoverre een element in de visual tree deze events afvuurt zal de ccSudoku9x9 control aan de bubbling- en tunneling fases deelnemen;
  - gebruik *e.OriginalSource* om te achterhalen welke control het event veroorzaakte.

jouwdatwerkt!!!

7.5 Async methodes: TaskCompletionSource

Indien opfrissing omtrent await/async nodig verwijs ik naar de Random topics: awaitable methodes.
Zoals in het hoofdstuk omtrent delegates werd aangekaart kan met behulp van de klasse TaskCompletionSource een awaitable methode ontwikkeld worden.

Het .NET framework 4.5 bevat veel awaitable methodes, herkenbaar aan de suffix Async (zie ook Random topics: awaitable methodes). Om dergelijke methode asynchroon uit te voeren, en de rest van de routine definitie in een callback methode voor die oproep te plaatsen volstaat het om await voor de methode oproep te plaatsen. Meer nog, de rest code volgend op de awaited

oproep wordt automatisch terug op de huidige thread geplaatst (meestal de UI-thread).

Het gebruik van een awaitable methode is gemakkelijk, het opzetten van een awaitable methode is dit minder. De volgende tekst beschrijft hoe een awaitable methode kan aangemaakt worden.

7.5.1 Task- klasses

Een awaitable methode heeft *altijd* een Task- object terug. Indien het return type van een synchrone oproep *void* is, is het return type van de asynchrone versie *Task*. Indien het return type van de synchrone uitvoering T was geweest zullen we $Task< T >$ als resultaat type gebruiken van de asynchrone oproep.

Indien de oproep wordt *awaited* mag het resultaat van de oproep (te vinden in de *Result*-paramter van de Task) als resultaat van de await gebruikt worden (de compiler lost dit op):

```
Boolean gevonden = await ZoekOplossingAsync(oplossing,0,mogelijkeKolommen);
if (gevonden)
    ToonOplossing(oplossing);
```

Figuur 7.5: Awaiting een oproep

Zowel de NQueens- als Sudoku oplosser geeft als resultaat een Boolean terug. Het resultaat type van de Async- versie zal dan ook $Task< Boolean >$ zijn.

7.5.2 TaskCompletionSource- klasses

De *await*- infrastructuur dient geïnformeerd te worden omtrent het verloop van de asynchrone uitvoering: voltooiing, onderbreking, voortgang en dergelijke meer zijn belangrijk. Omdat deze informatie niet enkel via een callback kan gecommuniceerd worden werd een nieuwe generische klasse geïntroduceerd: *TaskCompletionSource*:

- de informatie omtrent de uitvoering van een asynchrone taak wordt verzameld in een TaskCompletionSource- object;
- dergelijk object wordt door ons programma als field gedefinieerd zodat het kan gemanipuleerd worden door onze routines;

```
private void ZoekOplossingCompleted(IAsyncResult ar)
    {
    Cel[] cellen = ar.AsyncState as Cel[];
    if (cellen[cellen.Length -1] != null) //er was e
        _tcsOplossing.SetResult(true);
    else
        _tcsOplossing.SetResult(false);
    }
```

Figuur 7.6: TaskCompletionSource: SetResult

- de start van onze asynhrone methode initialiseert dit field, start het asynchrone proces (met klassieke callback) en geeft de Task- property van dit object als resultaat terug;
- het is nu de bedoeling dat onze asynchrone methode (of zijn callback) na uitvoering dit object als completed of cancelled gaat instellen. De compiler heeft code gegenereerd die hierop inpikt om dan de logica die wachtte verder uit te voeren.

Wellicht kan een codevoorbeeld dit verduidelijken:

```
private delegate Boolean ZoekOplossingDelegate(Cel[] vorigeZetten, int
private TaskCompletionSource<Boolean> _tcsOplossing;
private Task<Boolean> ZoekOplossingAsync(Cel[] vorigeZetten, int huidi
    {
    _tcsOplossing = new TaskCompletionSource<Boolean>();

    ZoekOplossingDelegate del = new ZoekOplossingDelegate(ZoekOplossin
    del.BeginInvoke(vorigeZetten,huidigeRij,mogelijkeKolommen,ZoekOplo

    return _tcsOplossing.Task;
    }
```

Figuur 7.7: TaskCompletionSource demo

- ZoekOplossingDelegate is de delegate klasse om het zoeken asynchroon te programmeren;
- tcsOplossing is het resultaat van de awaitable methode. Ze wordt als field gedefinieerd zodat mijn logica deze kan aansturen;
- ZoekOplossingAsync is de awaitable methode (herkenbaar aan het return-type en de suffix van de methode). Deze methode zal intern een asynchrone oproep starten en geeft onmiddelijk een Task-object terug. De compiler genereert code die in de oproeper de rest van de logica uitvoert nadat deze task voltooid wordt. Het voltooien van de Task gebeurt door het resultaat in te stellen op het *TaskCompletionSource*- object waar de Task onderdeel van uitmaakt;

7.6 Het MiniMax algoritme (geen examen stof)

Backtracking wordt onder andere gebruikt als Artificiële Intelligentie algoritme bij het ontwikkelen van (turn-based) games. Het *MiniMax* algoritme is hiervan een voorbeeld. De Engelstalige wikipedia-entry voor MiniMax vind ik inhoudelijk sterker dan de Nederlandstalige.

7.6.1 MiniMax benadering voor vier op een rij

Het MiniMax algoritme realiseert volgende doelstelling: indien de speler aan zet niet wint met zijn volgende zet, dan minimaliseert hij de kansen van de andere speler. De andere speler zal bij het overlopen van zijn eigen mogelijkheden, indien hij niet direct wint, veronderstellen dat zijn tegenstander (speler 1) geen fouten maakt, en diens score maximaliseren, vandaar de naam van het algoritme. Omdat we bordsituaties met elkaar vergelijken (om een beste of slechtste situatie te kunnen kiezen) zullen we een BordWaarde- functie nodig hebben die de waarde van een bordsituatie weergeeft.

lol, dat werkt!!!

7.6.1.1 BordWaarde

- de parameters zijn die waardes nodig om een bordsituatie te beschrijven;
- het resultaat is een numerieke waarde, zodat vergelijkingen mogelijk worden:
  - +1: indien de speler aan zet uitspeelt;
  - -1: indien de tegenspeler uitspeelt;
  - 0: anders.
- de BordWaarde- functie zal intens gebruikt worden: elke mogelijke stap zal geëvalueerd worden door deze functie. Het is dan ook belangrijk om deze functie zo efficient mogelijk te maken.

Het is contra- intuitief om de bord- waarde te beperken tot 3 waardes: winst, verlies of onbeslist. Immers, wanneer we zelf vier op een rij spelen streven we bijvoorbeeld naar een situatie waar twee aansluitende rijen in een kolom tot winst leiden (zie kolom 3 in onderstaande figuur). In onze beoordeling van een bord is dit duidelijk een zeer goede situatie. MiniMax gewijs zal de AI, indien hij drie zetten vooruit *denkt* (rekent) ook merken dat in elke mogelijke invulling van een dergelijke situatie we een winnende zet kunnen realiseren, wat van dergelijke bord- situatie een winnende situatie maakt (+1). In onderstaande figuur werd de situatie in kolom 3 hierdoor als winnend herkent (bij de vorige zet) omdat de AI minstens drie zetten vooruit denkt:

Figuur 7.8: 4 op een rij: bordwaarde

7.6.1.2 BesteZet

De functie BesteZet berekent voor een gegeven bord situatie de best volgende zet voor een speler. In een vier op een rij context is het resultaat van de oproep een kolom nummer (int).

- het resultaat van de oproep is het kolomnr waarin de speler zijn fiche moet plaatsen. De parameters van de oproep worden bepaald door de nood van de hieronder geformuleerde logica;
- één van de argumenten van de oproep, *depth* geeft aan hoeveel stappen de computer vooruit mag denken. Hoe meer stappen de computer vooruit denkt, hoe langer de berekening zal duren.
- de speler aan zet (de maximizing speler) overloopt elke mogelijke kolom en voert volgende zaken uit:
  - plaats een fiche in deze kolom;
  - indien vier op een rij werd gevonden hebben we een winnende kolom: *return kolomnr*
  - indien geen directe winst wordt de waarde van deze kolom bepaalt als MiniMax(maximizingSpeler, minimizingSpeler, depth -1, ..).
  - indien de MiniMax- waarde kleiner is dan het huidige minimum wordt deze zet het nieuwe minimum (bewaar zowel de kolom als het minimax resultaat);
  - verwijder deze fiche terug
- de kolom met de kleinste MiniMax waarde (de slechtst mogelijke zet voor de tegenstander) wordt als resultaat teruggegeven.

Om wat random zetten aan het algoritme toe te voegen kunnen we volgende zaken aanpassen:

- we houden niet enkel de eerste beste zet bij, maar alle zetten die de huidige beste waarde als resultaat hebben;
- bij het teruggeven van de beste kolomwaarde wordt een random kolomnummer uit de lijst gekozen.

lol, dat werkt!!!
Jef Dacks

7.6.1.3 MiniMax methode

De Engelstalige wikipedia-entry voor MiniMax beschrijft het MiniMax algoritme als volgt:

```
function minimax(node, depth, maximizingPlayer)
    if depth = 0 or node is a terminal node
        return the heuristic value of node
    if maximizingPlayer
        bestValue := -∞
        for each child of node
            val := minimax(child, depth - 1, FALSE))
            bestValue := max(bestValue, val);
        return bestValue
    else
        bestValue := +∞
        for each child of node
            val := minimax(child, depth - 1, TRUE))
            bestValue := min(bestValue, val);
        return bestValue
```

Figuur 7.9: Minimax algoritme

- een *terminal node* is een zet die het spel afrondt: één van beide spelers realiseert vier op een rij. We veronderstellen dat enkel de speler die nu een fiche plaatst kan uitspelen, en dat zijn huidige zet deel uitmaakt van de oplossing (dat wil zeggen: we veronderstellen dat er zijn geen andere vier op een rij oplossingen zijn).
- u moet weten wie de maximizing speler is opdat u weet of de speler aan zet de mogelijkheden van zijn tegenstrander minimaliseert of maximaliseert;
- die MiniMax algoritme overloopt *alle* mogelijke zetten (tot eindspelsituaties) voor beide spelers, wat in veel spelsituaties een gigantisch aantal mogelijkheden betekent. Het beperken van die set mogelijkheden, zonder de beste oplossing te verliezen is dan ook zeer wenselijk.

7.6.2 Minimax optimalisaties

7.6.2.1 Alfa-Beta pruning

Indien u het MiniMax algoritme correct kon implementeren (geen onderdeel van de cursus), dan zal u wellicht geïnteresseerd zijn in Alfa-Beta pruning voor de MiniMax zoek boom. De uitleg en het bijhorende voorbeeld onderaan de webpagina kan ik niet verbeteren.

Een nadeel van maximaal pruning is het verliezen van de alternatieve beste zetten. Dit kan als gevolg hebben dat de AI zijn zetten nogal éénzijdig in het speelveld plaatst: indien de kolommen van links naar rechts worden doorgelopen zullen de eerste beste oplossingen zich altijd links ten opzichte van hun alternatieven bevinden. Omdat enkel de eerste beste oplossing wordt weerhouden wordt het speelveld vooral links ingevuld.

7.6.2.2 Multi- threading

Op een multi core toestel zal het single- threaded berekenen van de beste zet geen optimaal gebruik maken van de beschikbare rekenkracht. Het op een aparte thread plaatsen van de BesteZet berekening maakt deze berekening zelf niet multi threaded. Het multithreaded maken van vier op een rij lijkt voor de hand liggend, maar moet ik zelf nog eens uitwerken. Bij het multithreaded maken moet rekening worden gehouden met volgende elementen:

- de verschillende threads mogen elkaars situatie niet beïnvloeden. Elke variabele die gewijzigd wordt moet dan ook een copy van de oorspronkelijke waarde zijn.
  - veronderstel dat elke kolom-invulling in een aparte thread wordt gestart. Voor de eerste zet van de eerste speler worden dan onmiddellijk de acht eerste kolommen ingevuld. Indien ze op een gemeenschappelijk veld worden geplaatst heb ik onmiddellijk *acht* op een rij gerealiseerd. Bovenstaande redenering is van toepassing op *elke* variabele die door een thread zal worden aangepast;
  - bij het multithreaded maken van de oplossing worden verschillende deelzoekbomen parallel uitgevoerd. Dit conflicteert met de winst bekomen door de alfa- beta pruning waarbij een nieuwe deelboom pas wordt verwerkt nadat de beste oplossing uit de vorige deelboom werd berekend.
    - wellicht is het alfa-beta prunen van een multithreaded oplossing wel realiseerbaar, maar het is maar de vraag hoe groot de winst nog zal zijn. Indien de multithreaded oplossing pas even snel is als de pruned singled threaded oplossing hebben we gewoon resources onnodig verbruikt. Hier is nog ruimte voor wat onderzoek :).

Ook dat werkt!!!

7.7 Backtracking opgaves

- ontwikkel een N-queens oplosser;
  - Extended WPF Toolkit bevat tal van interessante WPF-controls, onder andere verschillende up/down controls. Wellicht kan u een aantal hiervan gebruiken bij het opzetten van uw oplossing;
  - voorzie een synchrone uitvoering, en demonstreer het blokkeren van de interface;
  - voorzie een asynchrone uitvoering (met cancel- mogelijkheid) en demonstreer de responsiveness van de interface (bijvoorbeeld door de cancel uitvoering);
  - voorzie een tweede asynchrone uitvoering die gebruikt maakt van het async/await mechanisme (zie cursustekst).
- ontwikkel een Sudoku oplosser.
  - voorzie een synchrone uitvoering. Het resultaat wordt zo snel bekomen dat een asynchrone uitvoering weinig zin heeft.

lol, dat werkt!!!

lol, dat werkt!!!
Jef Deeh

Hoofdstuk 8

Dependency properties

8.1 Dependency properties: duiding

In een klassieke object georiënteerde omgeving definieert en implementeert een klasse de eigenschappen van haar instanties. In .NET wordt hiertoe een property- syntax gebruikt (we spreken hier van een CLR-property), in JAVA gebruikt men getter- en setter- routines. Zo zal de Text- property van de klasse TextBox in de TextBox-klasse zelf worden geïmplementeerd.

WPF introduceert de extra mogelijkheid om een property, gedefinieerd in klasse A, te koppelen aan instanties van klasse B. Dit vereenvoudigt sterk het opzetten van nieuwe infrastructuren: indien een Grid-klasse nood heeft aan het toekennen van *row* en *column* waardes aan subcontrols, dan kan de *Grid* klasse deze properties zelf introduceren. Deze dependency properties spelen ook een cruciale rol in WPF's databinding en animatie.

In praktijk kan u het dependency property systeem zien als een dictionary waarin voor objecten extra waardes worden bijgehouden. De infrastructuur laat toe om bij het instellen van de waardes custom code uit te voeren.

Attached dependency properties zijn XAML-georiënteerde dependency properties: om in XAML bruikbaar te zijn moet een dependency property een CLR wrapper hebben (die at run-time trouwens gebypassed wordt ..).

8.1.1 Relevante namespaces

- *System.Windows*
- *System.ComponentModel*
- *System.Windows.Media*
- *System.Windows.Data*

lol, dat werkt!!

8.2 F1URL: een help infrastructuur

We zullen een F1-gebaseerd help systeem opzetten voor de UI-elementen in onze toepassing(en). Hiertoe wensen we de mogelijkheid te hebben om voor een willekeurige control een URL met de gewenste help bij te houden. Wanneer de gebruiker F1 klikt zal de webpagina van de control waarboven de cursor zich bevindt getoond worden. Indien de controls genest zijn zullen de eventuele verschillende pagina's getoond worden. Momenteel wordt in het .NET framework niet de mogelijkheid voorzien om declaratief een URL te koppelen aan een control (u kan natuurlijk zelf een dictionary systeem opzetten).

8.2.1 F1URL gebruik in XAML

```
<Window x:Class="URLDemo.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525"
        xmlns:src="clr-namespace:URLDemo"
        x:Name="window"
        src:F1URL.URL="abc"
        >
    <src:F1URL x:Name="grid">
        <Button Height="175" Name="btnOuter"
                HorizontalAlignment="Left" Margin="128,69,0,0"  VerticalAlignment="Top" Width="262"
                src:F1URL.URL="www.google.com" >
            <Button Content="Button" Height="67" HorizontalAlignment="Left"
                    x:Name="btnInner" VerticalAlignment="Top" Width="121"
                src:F1URL.URL="http://www.howest.be" />
        </Button>
    </src:F1URL>
</Window>
```

Figuur 8.1: Attached dependency property in XAML

In bovenstaande XAML-code zijn volgende elementen van belang:

- de namespace *src* hebben we zelf toegevoegd (gebruik de intellisense!);
- *src:F1URL* is een klasse die erft van *Grid*. De controls waarop de F1URL benadering moet werken zullen binnen dergelijke grid geplaatst worden (de *F1URL*- control zal het *KeyUp* event opvangen voor alle geneste subcontrols);
- de (attached) dependency property *F1URL.URL* wordt ingesteld voor het window en twee (geneste) buttons. De geneste buttons introduceren onmiddellijk een extra complexiteit: indien we F1 drukken als de inner button geselecteerd is, wensen we dan de help van de inner of van de outer button?

Wanneer we de Button- klasse erop nazien merken we dat er geen URL-property gekend is. Hoe werd het dan mogelijk om deze property toch in te stellen?

8.2.2 F1URL definitie

```
public class F1URL : System.Windows.Controls.Grid
    {
    public static readonly DependencyProperty URLProperty =
        DependencyProperty.RegisterAttached("URL", typeof(string), typeof(F1URL), new UIPropertyMetadata(null));

    //[System.Windows.AttachedPropertyBrowsableForType(typeof(UIElement))]
    [System.ComponentModel.Browsable(true)]
    public static string GetURL(UIElement target)
        {
        return (string)target.GetValue(F1URL.URLProperty);
        }

    public static void SetURL(UIElement target, string value)
        {
        target.SetValue(F1URL.URLProperty, value);
        }
```

Figuur 8.2: Attached dependency property definitie ($C^{\#}$)

lol, dat werkt!!!

Bemerk in bovenstaande code volgende zaken:

- een dependency property is een *static* field van het type *DependencyProperty*. De naam van het field eindigt met het woord *Property*;
- een nieuwe dependency property instantie wordt aangemaakt door middel van een *register(Attached)*- oproep met volgende argumenten:
  - de naam in de registratie is identiek aan de fieldnaam, met weglating van de suffix *Property*;
  - het property type definieert het type van de property (een tekst- property heeft hiervoor als parameter *typeof(string)*);
  - het ownertype definieert de klasse die als prefix voor de property zal gebruikt worden: *F1URL*.URL (typeof(F1URL) is hierin het owner type);
  - extra informatie omtrent het verwerken van de property waardes.
- we voorzien de dependency property van (static) Get- en Set- wrapper routines (de hoofdletters *G* en *S* zijn belangrijk!). De Get- en Set- namen zijn (op de drie-letter prefix na) identiek aan de naam gebruikt in de register methode. Zonder deze wrappers is de dependency property niet gekend in XAML.
  - merkwaardig genoeg zal XAML de wrappers volledig bypassen, zoals beschreven in XAML Loading and Dependency Properties;
- indien we de Get- wrapper voorzien van een *Browsable*- attribuut zal deze dependency property ook getoond worden in de property list in Blend (in Visual Studio 2010 wordt de property getoond indien hij in de XAML-code aanwezig is);
- de snippet *propdp* is standaard in Visual Studio aanwezig om de dependency property definiërende code te genereren(*propa* voor attached dependency properties).

8.2.3 F1URL infrastructuur

We definieerden de dependency property *F1URL.URL* om een help infrastructuur op te zetten. Indien een gebruiker F1 duwt wensen we een help-pagina te tonen. Om het key-up event van een willekeurige control op te vangen gaan we als volgt te werk:

- we maken een eigen Grid-klasse versie waarbinnen alle controls die F1URL-ondersteuning nodig hebben geplaatst zullen worden;
  - de hier voorgestelde oplossing werkt enkel voor controls binnen onze eigen Grid-klasse. Deze beperking is overbodig wanneer we echte behaviors zullen ontwikkelen;
  - de huidige opzet is geschikt om bubbling (routed) events toe te lichten;
- de container control zal het bubbling *keyup*-event opvangen en actie ondernemen om de nodige help-topics te tonen:

```
public class F1URL : System.Windows.Controls.Grid
    {
    "dep prop URL"
    public F1URL() : base()
        {
        this.KeyUp += new System.Windows.Input.KeyEventHandler(URLF1_KeyUp);
        }

    void URLF1_KeyUp(object sender, System.Windows.Input.KeyEventArgs e)
        {
        if (e.Key == System.Windows.Input.Key.F1)
            verwerkF1(e.OriginalSource as FrameworkElement);
        }

    private void verwerkF1(FrameworkElement fwe)
        {
        if (fwe == null) return;
        string naam = fwe.Name;
        string url = (string) fwe.GetValue(F1URL.URLProperty);
        if (url != null)
            toonURL(url,naam);
        verwerkF1(fwe.Parent as FrameworkElement);
        }

    private void toonURL(string URL,string naam)
        {
        try
            {
            System.Diagnostics.Process.Start(URL);
            }
        catch (Exception ex)
            {
            }
        }
    }
```

Figuur 8.3: F1 infrastructuur ($C^{\#}$)

- de naam van het frameworkelement (een control parent klasse) is aanwezig omwille van debug-redenen;
- de *Process*-klasse start automatisch uw favoriete browser (bemerk dat *Process* ook het geschikte programma zal starten indien het argument geen URL zou zijn);

- de infrastructuur verwerkt alle geneste controls.

Uiteraard is deze eenvoudige benadering niet helemaal naar wens: zo willen we help kunnen opvragen omtrent een knop zonder(!) dat we deze dienen aan te klikken: het aanklikken start namelijk het proces waarvoor u de help wenst op te vragen, dit is zeker niet wenselijk. De infrastructuur moet rekening houden met de positie van de cursor en niet met de geselecteerde control. De lesdemo illustreert dit en maakt hiertoe gebruik van hit- testing om de control onder de cursor op te zoeken.

8.3 Dependency properties: technisch

- een dependency property is een static DependencyProperty- field in een klasse;
- een dependency property heeft een *owner type*. Dit is niet noodzakelijk de klasse waarbinnen de property wordt gedefinieerd. Een dependency property kan trouwens meerdere owner types hebben (we maken dan gebruik van de addOwner methode van de DependencyProperty klasse).
- een attached dependency property is een dependency property bruikbaar via XAML. Opdat XAML de property zou kennen moeten de naamgevingsregels gevolgd worden!
- bij het registreren van een dependencyproperty kan optioneel extra informatie worden meegegeven:
  - een default waarde;
  - verschillende routines (allen in de vorm van delegates):
    - propertyChangedCallback: wordt uitgevoerd wanneer de property waarde wijzigt (niet noodzakelijk bij elke toekenning!);
    - coerceValueCallback: wordt gebruikt om een eventuele nieuwe waarde om te vormen tot een andere waarde;
    - validateValueCallback: valideert de geldigheid van de nieuwe waarde voor deze klasse (niet in combinatie met de andere state elementen van het target object waaraan de value wordt toegekend). Het boolse resultaat wordt onmiddellijk gebruikt in de XAML-editor om de eventuele incorrecte waarde aan te geven. Deze routine wordt uitgevoerd *na* de coerce- routine;
- het definiëren van een dependency property is niet voldoende om deze property voor een object te kunnen instellen: dependencyproperties zijn enkel instelbaar op instanties van klasses die erven van DependencyObject!

lol, dat werkt!!!

Het is een aanrader om als delegate oefening eens een uitgewerkte URL- attached dependency property aan te maken met volgende elementen:

- een URL is slechts geldig indien hij start met *http://* (hoofdletterongevoelig);
- voor elke TextBox control zal de url *http://www.howest.be* worden omgevormd tot *http://www.nmct.be*.

Stel in XAML-code een aantal URL's in om uw implementatie te testen. Voorzie in uw testen volgende situaties:

- voorzie een URL-waarde die niet start met de gevraagde prefix, en observeer het resultaat in XAML en bij het opstarten;
- voorzie een textbox van een om te vormen URL;
- pas de coerce routine aan zodat de nieuwe waarde niet voldoet aan de validate routine en observeer het resultaat.

8.4 Dependency property XAML binding

Via XAML-code kunnen we enkel dependency properties *binden*. Indien we proberen een binding op te zetten voor een gewone (CLR) property, dan krijgen we bij uitvoering volgende foutmelding:

Figuur 8.4: CLR binding error

8.5 Dependency property syntax

Het instellen van een dependency property kan in $C^{\#}$op twee manieren gecodeerd worden:

- *target.setValue(ownertype.depprop, value)*: het object gebruikt de geërfde methode setValue om een dependencyproperty waarde te bewaren;
- *ownertype.Setdepprop(target,value)*: de static setter routine wordt gebruikt;

```
anWidth.SetValue(Storyboard.TargetNameProperty,btnInner.Name);
anWidth.SetValue(Storyboard.TargetPropertyProperty, new PropertyPath( Button.WidthProperty));
```

Figuur 8.5: Dependency property syntax 1 ($C^{\#}$)

```
Storyboard.SetTargetName(anWidth, btnInner.Name);
Storyboard.SetTargetProperty(anWidth, new PropertyPath(Button.WidthProperty));
```

Figuur 8.6: Dependency property syntax 2 ($C^{\#}$)

Er bestaan uiteraard gelijkaardige notaties voor het opvragen van de dependency property. Beide notaties komen aan bod gedurende het bespreken van de animaties.

8.6 Dependency properties en animaties

WPF ondersteunt een animatie infrastructuur (niet de topic hier) gebaseerd op dependency properties: het wijzigen van een dependency property met impact op de visualisatie zal deze visualisatie onmiddellijk wijzigen. Daarbovenop komen een aantal (animatie-) klasses die gebruikt kunnen worden om waardes van start- waarde naar een eind- waarde te laten evolueren gedurende een tijdsinterval (met easing). In combinatie met Blend maakt dit het vlot opzetten van tijdlijn animaties en storyboards mogelijk.

In dit cursusdeel zullen we de animaties implementeren door middel van $C^{\#}$- code. Door Blend gegenereerde code is XAML code. Het is *niet* de bedoeling om een impliciete voorkeur te laten blijken.

lol, dat werkt!!!

8.6.1 DoubleAnimation

De breedte van een control is een double waarde. De DoubleAnimation klasse voorziet de mogelijkheid om een double waarde geleidelijk te evolueren tusen een start- en eindwaarde. Indien men dan de breedte- eigenschap van een control koppelt aan deze double- evolutie bekomt men een animatie:

```
private void animeerBreedte()
    {
    DoubleAnimation anWidth = new DoubleAnimation(btnInner.Width, btnInner.Width * 1.1,
        new Duration(new TimeSpan(0, 0, 2)));
    anWidth.SetValue(Storyboard.TargetNameProperty,btnInner.Name);
    anWidth.SetValue(Storyboard.TargetPropertyProperty, new PropertyPath( Button.WidthProperty));
    anWidth.AutoReverse = true;
    anWidth.RepeatBehavior = RepeatBehavior.Forever;

    Storyboard sb = new Storyboard();
    sb.Children.Add(anWidth);
    sb.Begin(this);
    }
```

Figuur 8.7: Breedte animatie ($C^{\#}$)

Op basis van bovenstaande code formuleren we volgende opmerkingen:

- de DoubleAnimation instantie zelf weet niet op welke eigenschap van welk object ze zal worden toegepast;
- het koppelen van een DoubleAnimation aan een eigenschap van een control gebeurt door middel van twee dependency properties, gedefinieerd in de StoryBoard klasse:
  - *StoryBoard.TargetNameProperty* wordt gebruikt om de naam te bepalen van het element (de control) waarvan een eigenschap geanimieerd zal worden;
  - *StoryBoard.TargetPropertyProperty* wordt gebruikt om de dependencyproperty die geanimeerd zal worden te identificeren op basis van zijn PropertyPath.
- animaties worden (hier) als onderdeel van een StoryBoard uitgevoerd;
- het starten van een StoryBoard heeft nood aan een visueel element waarbinnen de te animeren controls worden gezocht.
  - in *WinRT*-toepassingen is het niet nodig de *Begin*- methode van een *StoryBoard* van parameters te voorzien.

8.6.2 ColorAnimation

Het animeren van de achtergrondkleur van een control gebeurt op gelijkaardige wijze maar is verschillend op volgende punten:

- de achtergrond van een control is een Brush (en geen Color!). Het animeren van de achtergrondkleur gebeurt dan ook door de kleur van een (SolidColor)Brush te animeren (we hebben een ColorAnimation- class, en geen SolidColorBrushAnimation class);
- het PropertyPath, nodig om een achtergrondkleur te animeren bestaat uit twee stukken: het eerste stuk kiest de Background property, het tweede stuk kiest daarvan de SolidColorBrush.Color property. De code zal enkel werken indien de background een SolidColorBrush waarde bevat. Indien dit niet het geval is (de achtergrond is niet ingesteld, of is een ander soort Brush), dan zal de achtergrond uiteraard niet geanimeerd worden (de gezochte eigenschap is niet aanwezig). Dit genereert trouwens *geen* uitvoeringsfout.

```
private void animeerBackground()
    {
    ColorAnimation anKleur = new ColorAnimation(Colors.Red,new Duration(new TimeSpan(0,0,2)));
    Storyboard.SetTargetName(anKleur, btnInner.Name);
    Storyboard.SetTargetProperty(anKleur,new PropertyPath("(Panel.Background).(SolidColorBrush.Color)"));
    anKleur.AutoReverse = true;
    anKleur.RepeatBehavior = RepeatBehavior.Forever;
    Storyboard sb = new Storyboard();
    sb.Children.Add(anKleur);
    sb.Begin(this);
    }
```

Figuur 8.8: Background animatie

lol, dat werkt!!!

8.6.3 Transformatie animaties

Het tranformeren van een figuur bestaat veelal uit drie onderdelen: een schaling, een rotatie en een skewing. Voor elk van deze onderdelen bestaat er een transformatie klasse. Verschillende transformaties kunnen worden gebundeld tot een transformatiegroep:

```
private void transformeer()
    {
    ScaleTransform scale = new ScaleTransform(0.75, 1.1);
    RotateTransform rot = new RotateTransform(90);
    SkewTransform skew = new SkewTransform(30, 30);
    TransformGroup tg = new TransformGroup();
    tg.Children.Add(scale);
    tg.Children.Add(rot);
    tg.Children.Add(skew);

    btnOuter.RenderTransform = tg;
    }
```

Figuur 8.9: Transformatie van een Button ($C^{\#}$)

lol, dat werkt!

- in veel gevallen is het gemakkelijker om een CompositeTransform te gebruiken in plaats van een transformatiegroep.
- bovenstaande code is nog geen animatie: er wordt een eenmalige transformatie toegepast op een control die daarna onveranderd blijft;

Een animatie zou er in bestaan om bijvoorbeeld de schaling en/of de draaihoek te animeren. Dit zijn properties van de zopas gedefinieerde objecten rot en scale. Deze objecten zijn niet bij naam gekend door de animatie infrastructuur (in de vorige animaties kon het te animeren object (een control) via zijn naam geïdentificeerd worden). Om properties van deze nieuwe objecten toch te kunnen animeren zullen we een naam voor deze objecten registreren. Na registratie kunnen ze via hun naam in een TargetNameProperty context gebruikt worden:

- in een *WinRT* toepassing kunnen we een object animeren zonder zijn naam te kennen door middel van de *TargetProperty*- dependency property van de StoryBoard klasse.

```
private void transformeer()
    {
    ScaleTransform scale = new ScaleTransform(0.75, 1.1);
    RotateTransform rot = new RotateTransform(90);
    SkewTransform skew = new SkewTransform(30, 30);
    TransformGroup tg = new TransformGroup();
    tg.Children.Add(scale);
    tg.Children.Add(rot);
    tg.Children.Add(skew);

    btnOuter.RenderTransform = tg;

    this.RegisterName("myrot", rot);
    this.RegisterName("myscale", scale);

    DoubleAnimation anAngle = new DoubleAnimation(rot.Angle, rot.Angle + 360,new Duration(new TimeSpan(0,0,4)));
    anAngle.SetValue(Storyboard.TargetNameProperty, "myrot");
    anAngle.SetValue(Storyboard.TargetPropertyProperty, new PropertyPath(RotateTransform.AngleProperty));
    anAngle.RepeatBehavior = RepeatBehavior.Forever;

    Storyboard sb = new Storyboard();
    sb.Children.Add(anAngle);
    sb.Begin(this);
    }
```

Figuur 8.10: Transformatie animatie ($C^{\#}$)

Jef Daels

8.6.4 Animaties en XAML

In veel gevallen is het gemakkelijker om animaties aan te maken met behulp van een design tool (*Blend*) dan via programmeer code. De door Blend gegenereerde code is XAML-code die buiten het bestek van dit hoofdstuk valt.

8.6.5 $C^{\#}$of XAML animaties?

Het animeren via code wordt naar mijn aanvoelen minder aangemoedigd dan het animeren via XAML (met behulp van tools zoals Blend). Toch meen ik dat een aantal situaties te herkennen zin waarbij $C^{\#}$- animatie een elegante oplossing kan bieden:

- animatie van dynamisch toegevoegde controls;
- animaties waarvoor de toe te passen waardes afhankelijk zijn van de run-time situatie en niet at-design time gekend zijn. Voorbeelden hiervan zijn:
  - het zweven van een fiche naar een speelveld op een spelbord;
  - het proportioneel verschuiven van elementen binnen een control, die niet altijd even groot moet zijn;

– een aantal basis- animaties (zoals bv. opacity) heb ik als extention method gedefinieerd op de control klasse. Het lijkt me dan ook weinig zinvol om deze animatie in XAML te definiëren.

Bovenstaande situaties zijn implementeerbaar via XAMl, maar behoeven dan meestal andere extra $C^{\#}$-code elementen zoals TypeConvertor of extra properties.

8.7 DependencyProperties in XAML: TypeConverters

Indien we in XAML een dependency property instellen moeten we ons beperken tot tekst. Dit is bruikbaar voor tekstwaardes, maar niet indien het resultaat type een niet-tekst type is. Om dit toch mogelijk te maken worden typeconvertoren gebruikt. Deze zijn in staat om op basis van een string een instantie van het property type te construeren (een vorm van deserialisatie). WPF bevat reeds een aantal typeconvertoren voor frequent voorkomende conversies (getallen, kleuren, ..) maar in sommige gevallen is het toch nodig om ook zelf een typeconverter te implementeren.

8.7.1 IValueConverter interface

Deze interface bevat een *Convert* en *ConvertBack* methode (in onderstaand voorbeeld is enkel de Convert van belang).

Het *ValueConversion* attribuut wordt gebruikt om het systeem te informeren omtrent de geïmplementeerde typeconversies. Onderstaande code codeert een converter die tekst omvormt tot een *LineairGradientBrush*. Er wordt verondersteld dat de twee kleurnamen in het Engels, en hoofdlettergevoelig geschreven worden:

```
[System.Windows.Data.ValueConversion(typeof(string),typeof(System.Windows.Media.LinearGradientBrush))]
public class LinearGradientConverter : System.Windows.Data.IValueConverter
    {
    public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
        {
        string tekst = value as string;
        if (tekst == null) return null;

        string[] kleuren = tekst.Split(new char[]{','});
        BrushConverter bc = new BrushConverter();
        Color kleur1 = ((SolidColorBrush) bc.ConvertFrom(kleuren[0])).Color;
        Color kleur2 = ((SolidColorBrush)bc.ConvertFrom(kleuren[1])).Color;
        return new LinearGradientBrush(kleur1, kleur2, 45);
        }

    public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
        {
        throw new NotImplementedException();
        }
    }
```

Figuur 8.11: ValueConverter definitie ($C^{\#}$)

l, dat werkt!!!

In XAML kunnen we dan volgende code noteren:

```
<Window.Resources>
    <src:LinearGradientConverter x:Key="lgc"></src:LinearGradientConverter>
    <s:String x:Key="redgreen">Red,Green</s:String>
</Window.Resources>
<src:F1URL x:Name="grid">
    <Button Height="129" Name="btnOuter"
            HorizontalAlignment="Left" Margin="128,69,0,0"  VerticalAlignment="Top" Width="262"
            src:F1URL.URL="http://www.google.com"
            Background="{Binding Source={StaticResource ResourceKey=redgreen}, Converter={StaticResource ResourceKey=lgc}}" >
    </Button>
```

Figuur 8.12: ValueConverter XAML ($C^{\#}$)

Hier wordt de achtergrond van een Button (een Brush dus) ingesteld door middel van een tekst (Red,Green) en een typeconverter. Het resultaat is een LineairGradientBrush.

Jammer in de XAML notatie is dat de tekst-waarde momenteel als static resource gedefinieerd wordt (misschien kan dit anders?)

lol, dat werkt!!!
Jef Peeels

8.7.2 Default dataconverters

Veelal lijkt de convertie van tekst in XAML naar een overeenkomstig object spontaan te gebeuren, zonder expliciete vermelding van de nodige type converter. Om dit te realiseren wordt een klasse voorzien van een default typeconverter (door middel van het TypeConverterAttribute attribuut). De typeconverter klasse wordt jammer genoeg anders opgebouwd dan de *ValueConverters* die eerder aan bod kwamen. Om een klasse als impliciete typeconverter te kunnen gebruiken gaan we als volgt te werk:

- de klasse die de conversie zal uitvoeren *moet* erven van de klasse *TypeConverter* (de interface *IValueConverter* is hier onbelangrijk);
- het is belangrijk om minstens twee methodes te overriden:
  - *CanConvertFrom* die default *false* terug geeft die aangeeft of het gevraagde brontype wel geschikt is om te converteren (in XAML is het brontype *String*);
  - *ConvertFrom* is de methode die de effectieve conversie uitvoert.

Wanneer een klasse een default typeconverter kent die tekst kan omvormen tot een instantie van de klasse, dan is het niet langer nodig om in de XAML-code een expliciete converter te voorzien.

8.7.2.1 TryParse

Wanneer ik een conversie van tekst naar een type X moet coderen zal ik het type X voorzien van een *TryParse* methode:

- het resultaat type is Boolean: true indien de conversie lukt, false anders;

- de eerste parameter bevat de tekstwaarde die geconverteerd moet worden;
- de tweede parameter (een referentie parameter) bevat het resultaat na de conversie (ongewijzigd indien de conversie niet lukt).

Het gebruik van *TryParse* is in .NET reeds op vele plaatsen terug te vinden (Integer, Double, DateTime, IPAdress, ..). Het implementeren van dergelijke methode is vooral een conventie, en zeker geen technische vereiste.

8.8 Dependency properties en OnRender

Indien een dependencyproperty belangrijk is voor het visuele aspect van de dependency objecten waarop de property wordt toegepast dan kan dit in de meta informatie worden opgenomen (bij het registreren van de dependency property). Indien dit het geval is zal het wijzigen van de dependencywaarde de *OnRender*- methode van het object waarvoor de dependecyproperty wijzigt triggeren. Dit op zijn beurt heeft als gevolg dat u het hertekenen niet moet oproepen via de propertychanged callback methode. Indien u dit wenst uit te pluizen (helemaal niet moeilijk) kan u zoeken op *FrameworkPropertyMetadata.AffectsRender* en *FrameworkPropertyMetadata.AffectsMeasure*.

lol, dat werkt!!!

8.9 Veel voorkomende fouten

- niet volgen van de naamgevings conventies (ook hoofdletters!);
- *null* als default waarde van een value type;

8.10 Extra lectuur

- Wie een interessant voorbeeld met vele depencendy properties wenst na te lezen verwijs ik graag door naar Render Text On A Path With WPF
- het aanmaken van een eigen dependency property wordt beschreven in Custom Dependency Properties
- een goede technische beschrijving omtrent attached dependency properties

8.11 Dependency property: opgaves

- animeer via code een control:
  - zijn afmetingen (breedte en grootte);
  - zijn achtergrondkleur;
  - roteer de control;
- maak een extention method waarmee u de opacity van een control (elke control) kan animeren: FadeInOut(double opacity);
- implementeer de *F1URL* infrastructuur en demonstreer dat deze werkt.
- voorzie de NQueens toepassing van een *Dimensie*- dependency properties, en bind deze aan de IntegerUpDown control zodat de code-behind leeg wordt;

```
<src:ccNQueens Grid.Row="1" x:Name="grdOplossing"
               src:ccNQueens.Dimensie="{Binding Value, ElementName=iudQueens}"  />
```

Figuur 8.13: XAML binding dependency property

lol, dat werkt niet
Jeroen Pijnenburg

Hoofdstuk 9

Templated controls

Eventuele oefening: een ccDatePicker

9.1 Templated controls: duiding

Een betrachting in moderne WPF toepassingen is het scheiden van logica en visualisatie (de wenselijkheid hiervan hangt af van de situatie). Dit wordt in veel gevallen gerealiseerd door het gebruik van XAML (usercontrols, templates, styles, ..) die gebonden wordt aan (dependency) properties gevonden in de logica (zie ook . Dit hoofdstuk behandelt extra mogelijkheden (bovenop het binden van properties) om interactie tussen beide lagen mogelijk te maken.

- de term *templated controls* duidt hier op die controls die in de visualisatie van het dataelement zijn opgenomen.
- dit hoofdstuk is geen pleidooi voor of tegen de scheiding tussen logica en view. Het al dan niet wenselijk zijn van die scheiding is projectafhankelijk, en de factoren zijn niet enkel puur technisch: vraag van de klant, huisstijl, beschikbare tijd, .. zijn evenzeer elementen die van belang zijn.

lol, dat werkt!!!

In dit hoofdstuk krijgt u zeer weinig code te zien. Het is dan ook de bedoeling om, met deze tekst als leidraad, zelf de code op te stellen: het is een goede oefening op delegates en dependency properties.

9.2 Relevante namespaces

9.3 CaptionControl

Wanneer we een WPF-toepassing maken is het niet zo eenvoudig om de titelbalk bovenaan een window aan te passen (u moet de windows themes wijzigen, wat zijn effect heeft op elke toepassing). In dit hoofdstuk zullen we een eigen control maken die als titelbalk kan dienen, maar waarvan we de eigenschappen (hoogte, lettertype, gebruikte images in de controlbox, ..) volledig zelf kunnen aansturen. Het is de bedoeling een custom control te ontwikkelen, zodat de visualisatie volledig wijzigbaar is. Het voorbeeld illustreert zowel het gebruik van PARTS als STATE's.

9.3.1 PARTS

9.3.1.1 Caption control via PARTS: inleiding

Als basis voor deze tekst gebruiken we MSDN: Creating a Control That Has a Customizable Appearance.

Wanneer u een custom control maakt geeft u de visualisatie van uw control uit handen: de gebruikende programmeur (designer) is in staat om de volledige visualisatie aan te passen. Indien u rekent op de aanwezigheid van een visueel element om een correcte verwerking te kunnen garanderen (onze CaptionControl verwacht dat er 'iets' is waarmee het window kan gesloten worden) is het wenselijk om dit aan de gebruiker (en zijn ontwikkel software: Blend of Visual Studio) duidelijk te maken. Dit gebeurt op twee manieren:

- PART_ prefix (ik vind dit niet terug als guideline, en wel een Pro WPF in VB 2010, pagina 567 die stelt dat deze naamgeving sinds de introductie van VSM verouderd is);
- *TemplatePart* attributen op de customcontrol.

In de *OnApplyTemplate* methode van de custom control kunnen we controls, aangemaakt door toepassing van de template opvragen en via events koppelen aan gewenst gedrag. U moet uw code voorzien op de eventualiteit dat de gebruiker de vereiste parts niet aanbood met de gewenste naam. In dat geval is het mogelijk dat een stuk functionaliteit niet langer aanwezig is, maar uw toepassing mag hierdoor natuurlijk geen fout veroorzaken!

Idpdat werkt!!!

Figuur 9.1: Window met eigen caption

9.3.1.2 Caption Control via PARTS: uitwerking

- maak een nieuw WPF project, en plaats de WindowStyle property van het aanwezige Window op *None*. Hierdoor zal het Window bij het opstarten geen caption meer vertonen. Het is de bedoeling om nu zelf een caption control te maken die dezelfde functionaliteit aanbiedt en beter aanpasbaar is;
  - hint: u kan dit venster nog steeds afsluiten met de *Alt-F4* toetscombinatie;

Figuur 9.2: Window met WindowStyle none

lol, dat werkt

- voeg via Visual Studio een *Custom Control ccCaption* toe aan het project. Hierdoor wordt een gepaste static constructor en een bijhorend theme aangemaakt. Save en compileer het project waarna u het in Blend opent;
- editeer de resource gevonden in *Generic.xaml*:
  - voorzie een grid met 4 kolommen;
  - de drie laatste kolommen worden gebruikt om de *minimize*, *maximize* en *close* functionaliteit aan te bieden. Omdat de te gebruiken images vierkant zijn zullen deze drie laatste kolommen net zo breed maken als ze hoog zijn (we binden de *Width* property aan de *ActualHeight* property van hun *Grid Ancestor*). De eerste kolom neemt de rest van de beschikbare plaats in beslag en zal de te tonen titel bevatten.
- plaats een Image in elk van de drie laatste kolommen, met als naam PART_Minimize, PART_Maximize, PART_Close;
- voorzie de custom control klasse van *TemplatePart* attributen die elk van deze drie images benoemen. Als type in het TemplatePart attribuut kiezen we niet *typeof(Image)* maar *typeof(UIElement)* om zo weinig mogelijk beperkingen aan de nodige elementen op te leggen.
  - plaats ook een control die de titel zal bevatten in de Grid, ook al zullen we deze in deze les verder niet verwerken;

- voorzie in de grid van het MainWindow een rij bovenaan waarbinnen de caption control getoond wordt. Verifieer dat de visualisatie klopt met de verwachtingen: indien u het Window resized moet ook de icoontjes resizen!
  - vergeet niet *VerticalAlignment* en *HorizontalAlignment* properties op Stretch te plaatsen en de *Stretch* property van de Images op *Fill*;
  - wellicht zal u de icoontjes die door Blend in de *Themes* folder werden geplaatst moeten verplaatsen naar een *images* folder die u zelf aanmaakt. U moet dan natuurlijk de XAML-verwijzing naar de ico-files ook aanpassen.

Om nu de gewenst acties aan de voorzien Images te koppelen gaan we als volgt te werk:

- override *OnApplyTemplate*. Gebruik *GetTemplateChild* om de UIElementen waaraan logica gekopeld moet worden (deze vermeldt in de *TemplatePart* attributen) op te halen;
  - koppel code aan het *MouseUp* event, wat overeenkomt met een *Click* event voor de gebruiker. Jammer genoeg is *Click* nog niet gekend op *UIElement*, reden waarom we *MouseUp* gebruiken.
  - om een window te sluiten moet u het window waarop de control staat kennen. U kan het window ophalen met een zeer eenvoudige recursieve routine die gebruik maakt van *VisualTreeHelper.GetParent*
    * voorzie een implementatie voor de drie bewerkingen: minimize, maximize en close.
    * u kan het Window waarbinnen een control zich bevind ook opvragen met behulp van een Window.GetWindow- oproep (met dank aan Bram Teirlynck)

lol, dat werkt!!!

Verifieer dat de gevraagde functionaliteit werkt.

9.3.1.3 Caption control via Parts: test en aanpassing

Gebruik Blend om in het formulier nieuwe template code voor de Caption control aan te maken: we vervangen minstens één van de Images door een *Button*. Merkwaardig genoeg zal die knop niet werken, ook al wordt de code doorlopen en het event gekoppeld: omdat sommige control-Types het MouseUp- event zelf afhandelen en *Handled* op *True* plaatsen wordt het event niet meer wordt doorgegeven. We zullen de event-koppeling op een andere manier moeten realiseren:

- kijk na hoe u met behulp van *AddHandler*- instructies een reeds afgehandeld event kan opvangen;
- verifieer dat het gebruik van de knop nu wel het beoogde effect heeft;
- pas ook de code voor de andere controls in de default template aan.

9.3.1.4 Caption Control via PARTS: evaluatie

- Deze manier van werken levert de client-ontwikkelaar flink wat vrijheid en geeft de control ontwikkelaar de kans om logica te koppelen aan (te verwachten aanwezige) visuele elementen;
- een nadeel is uiteraard de vereiste aanwezigheid van elementen met een specifieke naam, en het moeilijk koppelen van andere controls aan de gewenste logica (veronderstel dat we op de caption twee close- elementen wensen te plaatsen, terwijl er maar één template part hiervoor werd voorzien);
- deze methode is enkel toepasbaar indien u zelf de control ontwikkelt. Indien u een ander event dan dit dat default aan het Command is gebonden wenst op te vangen zal u gebruik moeten maken van de dependency properties benadering die wordt toegelicht in het volgende hoofdstuk.

9.4 Caption Control opgave

lol, dat werkt!!!

- maak een Caption Control zoals hiervoor beschreven en demonstreer zijn werking. Opdat u de images die u via Blend op uw controls plaats ook te zien zou krijgen gaat u best op de volgende manier te werk:
  - maak een *Images*- map in het project op hetzelfde niveau als de *themes*- map. De gemakkelijkste manier om dit te realiseren is rechts klikken in Blend en 'add folder' kiezen;
  - voeg de image- files toe aan deze directory (mbv de verkenner buiten Blend);
  - voeg deze image- files toe aan het project door in Blend rechts te klikken op de Images folder, kies *add existing item* en (multiselect!) kies de verschillende image-files. U zou deze nu ook in Blend moeten zien onder de Images folder;
  - stel deze images als source in voor de controls door deze in de keuzelijst (bij de source property) te kiezen.

met dank aan Sven Meesters (2011-2012)

9.5 States

Een visualisatie heeft hopelijk de mogelijkheid om verschillende statussen (states) van de getoonde informatie weer te geven: is een inputveld fout, teken dan een rode rand. Of zet de tekst zelf in het rood. Of plaats een uitroepteken achter het veld. Of doe een combinatie van voorgaande. Dit voorbeeld geeft aan dat de visualisatie, gekoppeld aan een bepaalde status van de gegevens, best niet bepaald wordt door het software gedeelte dat de fout detecteert, maar wel door het visualisatie gedeelte waarbinnen de fout moet benadrukt worden. We moeten ons natuurlijk niet beperken tot het al dan niet verkeerd zijn van informatie: zo zal een CheckBox het al dan niet aangevinkt zijn aangeven via een state (zie de volgende afbeelding), waarop de

visualisatie beslist hoe dit het best wordt weergegeven.

Figuur 9.3: Checkbox states visualisatie

Om dit gestructureerd mogelijk te maken zullen we als volgt te werk gaan:

- de custom control zal verschillende mogelijke states definiëren via TemplateVisualState-attributen op de control- klasse;
  - in de tekst staat beschreven hoe de onderlinge exclusiviteit gedefinieerd wordt;
  - deze states zijn trouwens ook gekend in Blend;
  - ook de .NET controls werken met dit states- systeem;

Iolf dat werkt!!!

Figuur 9.4: Checkbox states via Blend

- in de XAML-code kunnen we de overgang tussen de verschillende states geanimeerd visualiseren:

```
<Grid x:Name="grd">
    <VisualStateManager.VisualStateGroups>
        <VisualStateGroup x:Name="BezetState">
            <VisualState x:Name="Bezet">
                <Storyboard x:Name="sbBezet">
                    <ColorAnimation Duration="0:0:
                </Storyboard>
            </VisualState>
            <VisualState x:Name="OnBezet">
                <Storyboard>
                    <ColorAnimation Duration="0:0:
                </Storyboard>
            </VisualState>
        </VisualStateGroup>
```

Figuur 9.5: State animaties in XAML

  - u kan deze animaties perfect via Blend aanmaken door een storyboard aan te maken en overgangen tussen states aan te geven;
  - bemerk dat de start- en doel- status niet tot de animatie behoren;
  - indien u de XAML-code zelf codeert: u moet de VisualStateManager tag onmiddellijk onder de eerste control van de visualisatie coderen.

- langs de $C^{\#}$kant van de control moeten de state- overgangen aangegeven worden. Hiervoor moet de VisualStateManager.GotoState methode worden uitgevoerd. De MSDN-topic in de link beschrijft de verschillende parameters. Ook de derde parameter heeft zijn betekenis!
  - dergelijk gedrag leunt zeer sterk aan bij behaviors die later in de cursus aan bod komen.

lol, dat werkt!!

9.6 State opgave

Ontwikkel een verkeerslicht control volgens het PART- en State model. Het verkeerslicht wordt aangestuurd door op één van de lichten te klikken. Dit wijzigt de state van de control, waardoor een animatie dit licht ontsteekt en de andere lichten dooft (zie demo). Te volgen werkwijze:

- u *moet* werken met parts en states;
- u maakt de animatie om tussen de states te navigeren met behulp van Blend;

Figuur 9.6: State en Parte opgave: verkeerslicht

Jef Daels
lol, dat werkt!!!

Hoofdstuk 10

Behaviors

10.1 Behaviors: duiding

Door middel van dependency properties kunnen events op UIElementen worden afgehandeld. We spreken in dergelijk geval van *behaviors*. Oorspronkelijk geïntroduceerd in Blend 3.0 (en later opgenomen in .NET zelf) kan men behaviors gebruiken om een nieuw *gedrag* te koppelen aan uw UI. Indien de standaard aangeleverde behaviors niet voldoen kan u zelf een behavior ontwikkelen. In dit hoofdstuk zullen we Drag en Resize behaviors implementeren. Wellicht vind u op het internet vele gratis, interessante behaviors, naast deze die reeds in Blend worden aangeboden. Op het einde van dit hoofdstuk wordt een sectie gewijd aan het onthouden van de gewijzigde settings (afmetingen, posities, ...) in de application settings zodat bij het herstarten van de toepassing de aangepaste waardes toegepast worden. Ook een korte toelichting omtrent enumerations komt aan bod.

lol, dat werkt!!!

10.2 Relevante namespaces

- System.Windows.Interactivity

10.3 Drag behavior

Het verslepen van een UIElement is gebaseerd (en aangepast) op WPF tutorial .NET: Behaviors: we wensen de mogeljkheid te ontwikkelen om een control at run-time te verslepen. We zullen dezelfde behavior op drie manieren ontwikkelen:

- door middel van dependency properties: een goeie oefening en hopelijk verduidelijkend;
- via de Behavior die in Blend aanwezig is, maar niet bruikbaar voor Button- objecten;
- door een alternatieve Behavior te ontwikkelen, verder bouwend op de Blend *Behavior*-klasse.

10.3.1 Drag behavior: dependency property versie

We ontwikkelen een dependency property die, indien toegepast op een UIElement, toelaat dit te verslepen. Hiertoe moeten we drie events afhandelen:

- *MouseDownEvent*: we onthouden de huidige locatie (ten opzichte van de parent) en huidige muispositie en onthouden dat een *Drag*-beweging start (bMoveFlag wordt true);
- *MouseMoveEvent*: indien bMoveFlag true berekenen we de afstand die de muis aflegde sinds het MouseDownEvent en gebruiken deze afstand om de locatie van het UIElement met diezelfde afstand te wijzigen (het verplaatsen kan gebeuren door middel van een *TranslateTransform*;
- *MouseUpEvent*: we zetten de bMoveFlag op false, wat de eventuele verplaatsingen stopt.

Een beschrijving van de verschillende nodige elementen vind u in het tweede stuk van WPF tutorial .NET: Behaviors. Gelieve deze implementatie te realiseren en verifieer volgende zaken:

- de code werkt ook wanneer u een UIElement herhaalde keren (in dezelfde sessie) versleept;
- de code werkt ook voor een Button- control.

lol, dat werkt!!!
Jeff Davis

10.3.2 Drag behavior: Blend out of the box versie

In het voorgaande stukje werd een volledige eigen implementatie verzorgt, gebaseerd op dependency properties. Sinds Blend 3.0 worden een aantal behaviors standaard via Blend aangeboden, waaronder onder andere het verslepen van elementen. Het eerste stuk van hetzelfde artikel (WPF tutorial .NET: Behaviors) leert ons hoe we de Blend behavior *MouseDragElementBehavior* op een UIElement kunnen toepassen:

- volg de werkwijze van dit document en voorzie een control van de *MouseDragElementBehavior*-behavior;
- verifeer dat deze behavior niet werkt op een Button control.

10.3.3 Drag behavior: gespecialiseerde Blend versie

Omdat de standaard Blend-versie niet werkt voor een Button- control zullen we een eigen, Blend gebaseerde versie maken. We zullen hiervoor erven van *Behavior<T>* en de nodige methodes overriden. Het resultaat is een Behavior die ook vanuit Blend instelbaar is op een willekeurig UIElement. We gaan hiertoe als volgt te werk:

- de behavior werkt niet voor een Button, wellicht omdat de Button de klik- events zelf verwerkt en als handled markeert, waardoor ze niet langer *bubblen*. U kan een als *handled=true* bubbling event toch nog opvangen indien u gebruik maakt van de UIElement.AddHandler- instructie;
  - misschien is het ook wel de bedoeling van de Blend drag om niet te werken voor een Button. Immers, het draggen van een Button (mousedown, mousemove, mouseup) lijkt sterk op het klikken op de Button, en kan aanleiding geven tot verwarring;
- maak een nieuwe klasse die erft van *Behavior<UIElement>*. De *Behavior<T>* klasse wordt gevonden in een Blend libary. U kan deze library gemakkelijk terugvinden door middel van het vorige voorbeeld;
- in Behavior Generic Class wordt beschreven welke methodes overschreven kunnen worden. Eén van deze methodes is logisch sterk gelijkaardig aan de value changed handler die werd gecodeerd in de dependency property benadering. We gebruiken deze methode en copiëren de eerder ontwikkelde code in de nieuwe klasse (aangepast waar nodig);
- de nieuwe behavior kan vanuit Blend op een UIElement gesleept worden:
  - u selecteert de behavior (op basis van zijn naam) in het selectiewindow en sleept hem vanuit het selectie window op het UIElement;
  - het is niet mogelijke (in Blend 4) om de nieuwe behavior in de toolbox op te nemen, hij komt wel in de lijst van behaviors terecht.

lol, dat werkt!!!

Verifieer dat de nieuwe oplossing werkt op een Button element.

10.4 Grip behavior

Een *Grip* laat toe om een volledig window in plaats van een individuele control te verslepen. Eerder dan van scratch te herbeginnen en een bijna volledige copy te implementeren zullen we gebruik maken van overerving:

- erf van één van uw eigen drag- implementaties;
- in een volledige WPF omgeving zou het volstaan om de verondersteld aanwezige transformatie niet op het UIElement zelf toe te passen maar op het Window waarbinnen dit UIElement geplaatst werd (te vinden met de *VisualTreeHelper*). Jammer genoeg is een *Window* geen echt WPF object en kan men hierop geen RenderTransform instellen. We kunnen het window verplaatsen door zijn Top en Left properties aan te passen;

  - override de mousemove handler;
  - bereken, net zoals in de parent klasse de af te leggen afstand;
  - gebruik de *Top* en *Left* properties van het window om dit te verplaatsen;
  - het kan natuurlijk nodig zijn om een aantal elementen in de parent klasse niet langer als private maar nu als *virtual* te declareren.

10.5 Resize behavior

Een gelijkaardige (iets moeilijker) behavior is een *Resize*- behavior: het is de bedoeling om, wanneer op de rand van een control wordt geklikt, deze kan geresized worden. Hiervoor gaan we als volgt te werk:

- we erven van de Blend- Behavior classe en koppelen drie events;
- op het *MouseDownEvent* zullen we nakijken of er aan de rand wordt geklikt. Zoja wordt de cursor van het UIElement gewijzigd en onthouden we de mousedown positie;
- op het *MouseUpEvent* herzetten we de cursor tot zijn default;
- in het mousemove event verwerken we de verplaatsing op volgende wijze;
  - indien u moet resizen naar het oosten of naar onder volstaat het om de afmeting van de control aan te passen en de huidige muispositie als nieuwe mousedown te registereren;
    * indien u de mousedown positie niet vervangt wordt de afstand telkens ten opzicht van de originele positie genomen, zodat eerdere bewegingen terug geteld worden.
  - indien u naar het westen of noorden beweegt moet u de afmetingen van de control aanpassen *en* de control verplaatsen. Normaal gezien moet u de nieuwe muispositie niet registeren omdat die door het verplaatsen van de control terug samenvalt met de originele mousedown positie
    * net zoals in de Drag behavior oplossing is het verplaatsen van de control het gemakkelijkst te realiseren door middel van een translate transform.

Omdat we de *Width* en *Height* van een control manipuleren zal deze implementatie slechts werken indien deze reeds werden ingesteld. In een professionele implementatie is het nodig om, indien de Width of Height nog niet werden ingesteld, gebruik te maken van respectievelijk *ActualWidth* en *ActualHeight* om de berekeningen uit te voeren.

10.5.1 Alignment impact (Laurenz Demey 2010- 2011)

Wanneer de te resizen control *right aligned* is werkt bovenstaande code niet: de speciale afhandeling voor een resizen naar links (verbreden en verplaatsen) moet dan toegepast worden op de rechterkant (en mag niet toegepast worden op de linkerkant (analoog voor onder en bovenkant)

Jer Dgels
lol, dat werkt!!!

10.6 Behavior opgaves

- implementeer en demonstreer de Drag behavior als dependency property;
- illustreer de werking van de Drag behavior gevonden in Blend;
- implementeer en demonstreer de Drag behavior als ervend van *Behavior*<*T*>;
- implementeer een *Grip* behavior. Die laat toe om niet een control maar een volledig Window te verplaatsen. Illustreer de werking door deze toe te passen op de Caption controls gemaakt in Templated Controls;
- implementeer een resize behavior;
- plaats de Grip behavior op de Caption Control;
- aanrader: maak een Caption Control zoals deze in het vorige hoofdstuk, maar koppel de logica aan de UI-elementen door middel van dependency properties.
- een model examen opgave had kunnen zijn: pas de F1URL implementatie aan zodat het een behavior wordt:
  - u bent dan niet langer gebonden aan de eigen grid klasse waarbinnen de controls moeten geplaatst worden;
  - u hebt niet langer nood aan een hittest routine om te bepalen welke control de help opvraagt;
- een model examen opgave zou kunnen zijn: ontwikkel de *Behavior*- klasse zelf voor de Windows Store omgeving;

lol, dat werkt!!

10.7 WPF en application settings

Een interessante feature in de WinForms-designer die ontbreekt in WPF-toepassingen is het gebruik van application settings (instellingen) die UI-properties over sessies heen onthouden. Mits een weinig infrastructuur werken loopt dit echter wel los:

- voeg in Visual Studio aan de properties van uw toepassing de nodige settings manueel toe (kies een geschikte naam en datatype). De designer zal een application settings aanmaken met een static default instantie (u kan de code nakijken in de file *Settings.Designer.cs*.
- koppel de te onthouden WPF eigenschappen aan hun overeenkomstige setting:

```
Height="{Binding Source={x:Static src:Properties.Settings.Default}, Path=hoogte, Mode=TwoWay}"
```

Figuur 10.1: Height en applicationsettings

- anders dan in *VB.NET* worden in *C#*de application settings niet automatisch bewaard. Daarom zullen we in het *Closed* event van het Window de applicationsettings zelf bewaren: *JDADragBehavior.Properties.Settings.Default.Save();* (waarbij JDADragBehavior de naam van het project is).

lol, dat werkt!!!

10.8 Enumerations: getalwaardes en bewerkingen

In mijn eigen implementatie van de Resize behavior ontwikkelde ik en enumearation om de rand waarbij de cursor zich bevind bij te houden:

```
public enum RandPositie
    {
    Midden=0,
    Noord=1,
    Zuid=2,
    Oost=4,
    West=8
    }
```

Figuur 10.2: Enumeration definitie

lol, dat werkt!!!

Indien een variabele een combinatie van verschillende waardes moet bevatten (stel Noord en West) dan worden beide waades opgeteld met de + operator. Indien we willen testen of een waarde onder andere de Noord waarde bevat gebruiken we de & operator. Door de getalwaardes als machten van 2 te kiezen is elk getal slechts op exact één manier een combinatie van de voorgaande. Hierdoor is geen verwarring mogelijk.

- het getal 7 is met voorgaande definitie slechts mogelijk door combinatie van 1 (Noord), 2 (Zuid) en 3 (Oost);
- 7 & Zuid levert een waarde verschillend van 0 (midden);
- 7 & West lever 0 (midden) op.

Om de uniciteit van de combinaties te begrijpen kan een binaire invalshoek helpen:

- doordat de getallen machten van 2 zijn, zijn ze in een binaire voorstelling allemaal 1 gevolgd door een aantal nullen. Het combineren van verschillende enum-elementen zet de overeenkomstige 1-waardes actief (elke 1-waarde kan slechts door 1 enum waarde worden opgezet, gezien we geen dubbels toelaten). Het getal resultaat wordt voor het leesgemak in de code decimaal genoteerd;
- indien u de binaire voorstelling van een getal nakijkt kan u op basis van de 1-tjes nakijken welke enumwaardes deel uitmaken van dit getal. De & operator is uitermate geschikt om na te kijken of een specifieke enumwaarde (die maar één 1 bevat) deel uitmaakt van een gecombineerde set waardes.

10.9 Extra lectuur

- The Attached Behavior pattern: een theoretische duiding;
- Introduction to Attached Behaviors in WPF: een zeer eenvoudig en verklarend voorbeeld;
- WPF tutorial .NET: Behaviors: een mooi voorbeeld in twintig regels code, verifieer of het werkt bij herhaalde uitvoeringen van dezelfde sessie;
- Behavior Generic Class

Jef Daels
lol, dat werkt!!!

Hoofdstuk 11

iTextSharp

iText is een in België onwikkelde (open source) bibliotheek waarmee PDF-documenten gegenereerd en gemanipuleerd kunnen worden. Deze bibliotheek gebruik ik bijvoorbeeld om een watermerk in deze PDF te plaatsen.

lol, dat werkt!!!

11.1 Installatie

- start een nieuw WPF project;
- *Project.Manage NuGet Packages ..*;
- zoek op *iText* en installeer *iTextSharp* (de $C^{\#}$versie van iText);

11.2 PDF creatie (en opgaves)

Voor een eenvoudige tutorial omtrent het aanmaken van een nieuwe PDF verwijs ik graag naar Basic PDF Creation Using iTextSharp.

Een beetje rondzoeken op het internet helpt u wel verder omtrent een aantal topics. Zo moet u in uw test document volgende zaken realiseren:

- plaats een figuur in de pdf en herschaal deze tot 150x150;
- maak een titel paragraaf met een grotere fontsize en gecentreerd;
- maak een paragraaf die bestaat uit chunks die elk hun eigen kleur hebben;
- trek een lijn in een pdf. Let wel: het PDF coördinaten systeem is verschillend van dit dat u gewoon bent in een informatica omgeving:
  - de oorsprong is links onder op het blad, en niet links boven;
  - een grotere y- waarde ligt hoger op het blad, terwijl dit in een windows omgeving lager ligt.
- ...

11.3 Manipulatie van een bestaande PDF (en opgaves)

- maak een nieuw PDF document door een aantal PDF-documenten achter elkaar te plaatsen;
- lees een pdf document en plaats 4 logische pagina's op 1 fysisch blad;
- plaats een watermerk op elke even pagina van een document.

11.4 Tekenen (en opgaves)

Voorzie uw Sudoku oplossing van de mogelijkheid om een Sudoku te exporteren naar PDF:

Figuur 11.1: Sudoku in PDF

Let op volgende zaken:

- de opgave- getallen zijn anders gekleurd dan de oplossing- getallen;
- de 3x3- hokjes zijn van elkaar gescheiden door dikkere lijnen;

11.5 Toepassingsgebied voor iText

- *niet* als tekstverwerker;
- wel als rapport generator (veel veel sneller dan office activeX);

lol, dat werkt!!!

lol, dat werkt!!!
Jef Deeb

Hoofdstuk 12

SQLServer en .NET integratie

Relationele database laten meestal toe om logica in de vorm van stored procedures in de database op te nemen. Naast administratieve voordelen betreffende security en onderhoud is ook de potentiële performance winst een troef: berekeningen op de server dienen niet alle nodige gegevens over het netwerk naar de client te sturen, enkel het resultaat sturen is voldoende. Oorspronkelijk diende deze logica in een RDBMS specifieke, behoorlijk ouderwetse taal gedefinieerd te worden. Sinds enige jaren bestaat de mogelijkheid om de logica in uw favoriete programmeeromgeving te ontwikkelen en deze daarna in de server te plaatsen. Dit hoofdstuk behandelt het opnemen van .NET programmatuur in SQLServer.

lol, dat werkt!!

12.1 Nodige softwares

- Visual Studio 2012;
- SQL Server 2012;
- SQL Server Datatools 2012.

Installatie van de SQL Server Datatools maakt een nieuwe Visual Studio versie beschikbaar. Deze dient u te gebruiken voor deze les.

12.2 Kwadraten berekenen

Wellicht weinig zinvol in een database context, maar gemakkelijk om het principe uit te leggen is het aanmaken van een functie die bij oproep een tabel van kwadraten aanlevert. Indien u zelf meer voorbeelden wil zal Usage Scenarios and Examples for Common Language Runtime (CLR) Integration wellicht een goed startpunt zijn.

- start *SQL Server Data Tools 2012*, een *Visual Studio* oplossing;
- voeg aan de SQL Server object explorer uw lokale SQL Server 2012 installatie toe;
- voeg de database SQLNET toe. Dit is de database waarbinnen we onze test zullen uitvoeren;
- lees mij;
- *Create New Project* op de zonet gemaakte database, met als naam: *KwadraatDemo*;
- voeg de klasse *Kwadraat* toe:

Jef Daelt!!!

```
public class Kwadraat
    {
    private SqlInt64 _getal, _kwadraat;
    public Kwadraat(SqlInt64 getal, SqlInt64 kwadraat)
        {
        _getal = getal; _kwadraat = kwadraat;
        }
    [SqlFunction(FillRowMethodName = "VentileerRijen", TableDefinition = "getal bigint, kwadraat bigint")]
    public static IEnumerable MaakKwadraten(int maxgetal)
        {
        try
            {
            ArrayList kwadraten = new ArrayList();
            for (int i=1;i<=maxgetal;i++)
                kwadraten.Add(new Kwadraat(i, i*i));

            return kwadraten;
            }
        catch (Exception ex) { return null; }
        }
    public static void VentileerRijen(object okwad, out SqlInt64 getal, out SqlInt64 kwadraat)
        {
        Kwadraat kwad = okwad as Kwadraat;
        getal = kwad._getal;
        kwadraat = kwad._kwadraat;
        }
```

Figuur 12.1: Kwadraat code

- compileer en publish deze code;
  - voeg de nodige referenties aan het project toe;
  - bemerk dat SQLServer datatypes verschillend zijn van .NET datatypes;

  - bij het publishen moet u de nodige parameters voorzien om in de juiste database, op de juiste server te deployen;

- indien het publishen lukt zal u in de database onder de *table value functions* de routine *MaakKwadraten* terug vinden.

- u kan deze functie gebruiken om een tabel kwadraten te selecteren:

Figuur 12.2: Kwadraten selecteren

lol, dat we

Omtrent bovenstaande code verwijs ik graag naar het internet voor een eerste toetsting van uw vragen.

12.3 SQL Server tabellen raadplegen

Wanneer u in een klassieke toepassing SQL Server tabellen raadpleegt dient u een database connectie te configureren. Omdat we nu code ontwikkelen die in SQL Server opgenomen zijn en dus geconsulteerd worden via een bestaande connectie die we willen hergebruiken zal dit op een net iets andere manier gerealiseerd worden

- maak een tabel (vergeet de PK niet) in de SQLNET database en voorzie deze van een aantal rijen;

- lees mij;

- gebruik uw verzamelde kennis (deze les en vorige jaren) om een TVF te maken die iets uit die tabel ophaalt en terug geeft. Maak bijvoorbeeld een select met een where- conditie waarbij een parameter wordt meegegeven (zie demo).

De kans is groot dat dit niet onmiddellijk lukt, wat ons onmiddellijk de nood aan debugging demonstreert.

12.4 Debugging SQLServer

12.4.1 Debugging startend vanuit het Visual Studio project

Het opzetten van de debugging in SQLServer 2012 gebeurt net iets anders dan in de vorige SQL Server versies:

- in de *SQL Server Object Explorer* (dit is *niet* SQL Server Explorer) kan u door te rechtsklikken het debuggen aanzetten;
- in de properties van het project verifieert u in de *Debug* farde of de connection settings in orde zijn. Gebruik de edit- knop indien dit niet het geval zou zijn;
- voeg een script toe aan het project waarin u het te testen SQL- statement opneemt. Opnieuw in de debug farde stelt u het startup script in op dit script (er staat een browse knop helemaal rechts);
- plaats break points in uw .NET code en start het project. Indien alles goed verloopt komt u in een debug sessie terecht.

Het debuggen, interpreteren en oplossen is voor de lezer.

lol, dat werkt!!!

12.4.2 Debugging van een SQL Server call gestart buiten het project

Indien bijvoorbeeld in Management Studio software een query gestart wordt die code uitvoert waarvan wij het project hebben kunnen we deze uitvoering als volgt debuggen:

- open het Visual Studio project en attach aan *sqlservr.exe*;
  - *Tools.Attach to process ...*;
  - zet *show processes from all users* op omdat de sqlserver service wellicht niet met uw eigen userid gestart is;
  - bij het attachen krijgt u een beveiligings boodschap die u waarschuwt voor eventuele risico's;
  - dit attachen brengt uw project trouwens in een running state;
- voer een select statement uit die de code in uw Visual Studio project uitvoert. Indien u een breakpoint plaatste zal de uitvoering daar onderbroken worden en kan u uw gebruikelijke debug acties starten.

12.5 Theorie vragen

Met onderstaande vragen test u uw begrijpen van de software en krijgt u een idee omtrent de mogelijke detailvragen voor dit hoofdstuk.

- het SqlFunction- attribuut heeft een FillRowMethodName parameter. Deze wordt in onze klasse niet opgeroepen. Leg uit wat de functie van deze methode is.
- het SqlFunction- attribuut heeft een TableDefinition parameter. Welke zijn de mogelijke datatypes?
- de datatypes gebruikt in de TableDefinition parameter zijn niet identiek aan deze gebruikt in de FillRowMethode. Verklaar!
- opdat .NET programma's in SQLServer bruikbaar zouden zijn moet een setting worden ingesteld. Is deze setting een server, database of assembly instelling?
- opdat uw .NET programma's in SQLServer via Visual Studio debugbaar zouden zijn moet een setting worden ingesteld. Is deze setting een server, database of assembly instelling?

lol, dat werkt!!!

lol, dat werkt!!!
JeDaels

Hoofdstuk 13

Windows installer oplossingen

Om een desktop toepassing te verspreiden is het wenselijk om gebruik te maken van een MSI- bestand (Store toepassingen worden verspreid via de Store). Dergelijk installatiebestand ondersteunt een aantal standaard-acties zoals daar zijn: het voorzien van desktop- en start-icoontjes, het voorzien van een *uninstall* mogelijkheid, het automatisch mee- installeren van de nodige dll's en dergelijke meer.

Vanaf versie 2012 van Visual Studio verdwijnt de mogelijkheid om *Visual Studio setup-projecten* in een solution op te nemen. Deze feature werd vervangen door de introductie van *InstallShield Limited Edition for Visual Studio*, de gratis variant van de klassieker InstallShield. Het is belangrijk op te merken dat er ook andere al dan niet gratis softwares bestaan waarmee *MSI* pakketten kunnen ontwikkeld worden. In deze cursus komen het reeds vermeldde *InstallShield Limited Edition* en WiX (ontwikkeld door Microsoft) aan bod.

lol, dat werkt!!!

13.1 Demo programma

Ontwikkel een WPF-toepassing met volgende features:

- er worden twee assemblies aangeleverd: een .exe en een .dll;
- de UI van de .exe toont een knop die, wanneer hij wordt geklikt, een string uit de .dll ophaalt en toont in een MessageBox.
- we hebben een .ico file die we als desktop icoon wensen te gebruiken voor deze toepassing.

13.2 InstallShield Limited Edition

13.2.1 Download en installatie

Even Googlen en u vindt een gepaste download. Het resultaat van de installatie integreert netjes met Visual Studio waar een nieuw project type beschikbaar wordt gesteld onder *Other Project Types/Setup And Deployment*.

13.2.2 Aanmaken van een MSI-pakket

Het is de bedoeling dat de student zelf wat experimenteert met de software. U dient minstens volgende zaken te realiseren:

- maak een setup waarin minstens twee files voorkomen die geïnstalleerd moeten worden;
- wijzig de schermen van het installatie proces: gebruik images en wijzig de licentie tekst;
- de installatie moet een desktop-icon en een startmenu icon voorzien (het startmenu icon werkt momenteel niet onder windows 8).

13.2.3 Redistributables

Wellicht is uw software afhankelijk van andere softwares, zoals het .NET framework, SQLServer, .. . U kan deze softwares meenemen in het installatiepakket. Om één of andere reden lukt dit me niet voor het .NET framework bij het gebruik van de gratis InstallShield.

lol, dat werkt!!!
Jef Raels

13.2.4 Evaluatie InstallShield Limited Edition

Ik ben er niet wild van:

- na één jaar Windows 8 is het geïnstalleerde programma nog steeds niet op het start scherm te plaatsen;
- redistributables opnemen in het pakket (of deze automatisch door de client laten downloaden) lukt me niet. Misschien enkel in de betalende versies?
- een groot aantal features wordt in de gratis versie enkel vermeld maar kan pas gebruikt worden in de betalende versies:
  - het bepalen van dependencies gebeurt niet in de gratis versie;
  - setup- design;
  - ...

Deze software lijkt me bruikbaar in eenvoudige situaties, zoals een lescontext. Een voordeel is dan weer wel dat de wizard driven interface gemakkelijk in gebruik is. Wellicht is de betalende versie ook bruikbaar in complexere situaties.

13.3 WiX

Wix is een XML gestuurde software met een stijle leercurve. Een tutorial is dan wellicht ook op zijn plaats.

13.3.1 Download en installatie

Google .. .

13.3.2 Aanmaken van een MSI-pakket

Het is de bedoeling dat de student zelf wat experimenteert met de software. U dient minstens volgende zaken te realiseren:

- maak een setup waarin minstens twee files voorkomen die geïnstalleerd moeten worden;
- wijzig de schermen van het installatie proces: gebruik images en wijzig de licentie tekst;
- de installatie moet een desktop-icon en een startmenu icon voorzien (het startmenu icon werkt momenteel niet onder windows 8).

lol, dat werkt!!!

13.3.3 Redistributables

Wellicht is uw software afhankelijk van andere softwares, zoals het .NET framework, SQLServer, .. . U kan deze softwares meenemen in het installatiepakket. In een Wix- benadering moet u een extra *Bootstrapper Project* (dit is een speciaal project type) aanmaken. Dit project type specifieert een *Chain* van te installeren softwares. Uw eigen software (via het WiX installatie project) en de te installeren software (bijvoorbeeld het .NET framework) maken hier onderdeel van uit.

13.3.4 Evaluatie WiX

De leercurve van WiX is *veel* stijler dan deze van InstallShield. Toch verkies ik Wix boven InstallShield (een privé mening), wellicht om volgende redenen:

- de gratis versie is geen beperkte versie;
- de (complexe) XML- aansturing (en de online documentatie) geven me toch wel het idee dat ik de meeste situaties geconfigureerd kan krijgen, waar ik met de gratis InstallShield versie toch wel wat beperkingen wordt opgelegd.

13.4 Welke installer?

13.4.1 Vergelijking InstallShield en Wix

Indien geld geen probleem zou zijn verwacht ik een grote meerwaarde bij het gebruik van een betaalde InstallShield installatie. Indien enkel de gratis versies in overweging worden genomen verkies ik WiX omdat ik vrees dat de free InstallShield versie niet voor alles toereikend zal zijn, en ik op termijn dus toch WiX zal nodig hebben.

13.4.2 Vergeet niet:

Er zijn nog andere installatie softwares, zowel gratis als betalend, dit hoofdstuk heeft geenszins de bedoeling om een uitputtende vergelijking tussen de verschillende mogelijkheden te maken.

13.5 Voorbeeld labo examen opgave

- maak een setup voor uw sudoku programma, zowel met InstallShield als Wix;
- voorzie in de setup ook een PDF met een getekende Sudoko die mee geïnstalleerd wordt (de veronderstelde help).

lol, dat werkt!!
Jef Dekeyn

Hoofdstuk 14

AOP: Aspect Oriented programming

14.1 AOP duiding

Bij het ons welbekende programmeren is het gebruikelijk om programmalogica logisch op te delen in verschillende klasses en methodes: elke methode staat in voor de correcte afhandeling van een deelprobleem. Hoewel dit suggereert dat elk probleem op één welbepaalde plaats wordt afgehandeld zijn er *aspecten* van het programma die zich *niet* tot een welbepaalde plaats beperken: *logging*, *parameter validatie*, *authorisatie checks*, ... komen telkens opnieuw, op verschillende plaatsen terug. *Aspect Oriented Programming* biedt de mogelijkheid om de aspect gerelateerde code op één welbepaalde plaats te coderen:

lol, dat werkt!!!

.NET ondersteunt AOP door middel van de ContextBoundObject- klasse. Omwille van de beperktheid van deze oplossing zullen we voor deze lessen gebruik maken van *PostSharp*, hoewel dit natuurlijk niet het enige alternatief is. In Rating of Open Source AOP Frameworks in .NET worden een aantal frameworks met elkaar vergeleken. Recent werd *PostSharp* een commercieel produkt (het was oorspronkelijk gratis), reden waarom we gebruik maken van een trial versie.

Onderstaande afbeelding illustreert het beperken van een examenpunt tot een getal tussen nul en twintig, door middel van een attribuut. Het afdwingen van de parameter- beperking zal gebeuren door middel van het *OnEntry*- aspect.

```
public static void ingaveStudentModulePunt(string student, string module,[AOPIntegerRange(0,20)] int punt)
    {
    ////parameters testen
    //if (punt < 0) throw new Exception(punt + " is kleiner dan 0");
    //if (punt > 20) throw new Exception(punt + " is groter dan 20");

    Console.WriteLine(student + " haalde voor " + module + " " + punt + " punten.");
    }
```

Figuur 14.1: AOP: parameter annotatie

14.1.1 Relevante namespaces

- PostSharp.

14.2 PostSharp

14.2.1 PostSharp kennismaking: methode-entry teller

Indien u versie 2.5 installeert (Google) gaat u na installatie als volgt te werk:

- ontwikkel een Fibonacci oplossing
- use *Project.Manage NuGet packages* to add the *Postsharp-Aspects* package;
- maak een eigen klasse *clsEntryTeller* die erft van *PostSharp.Aspects.OnMethodBoundaryAspect*;
- override de *OnEntry* methode. Deze wordt uitgevoerd telkens een methode, voorzien van het *clsEntryTeller* attribuut wordt gestart (we zullen dit attribuut op alle methodes van een klasse toepassen door het attribuut op klasse niveau te definiëren);
- voorzie logica om de uitvoering van deze methode te registreren. Dit zal wellicht gebeuren door middel van een static field in *clsEntryTeller* dat voor elke methode het aantal uitvoering bijhoudt (gebruik een *Dictionary*?);
- voorzie logica (statisch?) om de informatie, verzameld in *clsEntryTeller*, in de console te dumpen.
- programmeer wat test- logica en voorzie de te tellen methodes van het *clsEntryTeller* attribuut. Dit kan gebeuren op verschillende manieren:
  - u definieert het attribuut voor de methodehoofding (bruikbaar voor een klein aantal methodehoofdingen);
  - u definieert het attribuut op klasse (of assembly)- niveau en vermeld (met behulp van reguliere expressies) welke methodes al dan niet van het attribuut moeten voorzien worden.
- test het resultaat.

lol, dat werkt!!!

PostSharp weeft IL-code rond de methodeoproepen waarop het *OnMethodBoundaryAspect* werd toegepast. De door PostSharp geïntroduceerde namen bevatten tekens die door ons bij het programmeren niet gebruikt kunnen worden. Dit vermijdt potentiële naamconflicten tussen onze code en de gegenereerde code. Zoals reeds werd vermeld in het hoofdstuk omtrent runtime compilatie wordt dit code-weven gestart door in het *MSBuild* proces in te grijpen.

ILSpy maakt duidelijk hoe de uiteindelijke gegenereerde IL code er uit ziet (voor de hieronder gegenereerde IL-code werd ook de *OnException* methode overriden, zodat het hier getoonde try-catch blok werd gegenereerd):

```
public class Wiskunde
{
    [System.Diagnostics.DebuggerNonUserCode, System.Runtime.CompilerServices.CompilerGenerated]
    internal sealed class <>z__Aspects...
    public static int Fibonacci(int i)
    {
        MethodExecutionArgs methodExecutionArgs = new MethodExecutionArgs(null, null);
        MethodExecutionArgs arg_13_0 = methodExecutionArgs;
        System.Reflection.MethodBase m = Wiskunde.<>z__Aspects.m3;
        arg_13_0.Method = m;
        Wiskunde.<>z__Aspects.a0.OnEntry(methodExecutionArgs);
        int result;
        try
        {
            int num;
            if (i <= 0)
            {
                num = 0;
            }
            else
            {
                if (i == 1)
                {
                    num = 1;
                }
                else
                {
                    num = Wiskunde.Fibonacci(i - 1) + Wiskunde.Fibonacci(i - 2);
                }
            }
            result = num;
        }
        catch (System.Exception CS$0$5__exception_58)
        {
            Wiskunde.<>z__Aspects.a0.OnException(methodExecutionArgs);
            throw;
        }
```

Figuur 14.2: PostSharp en ILSpy

14.2.2 PostSharp en parameter annotaties

Het codevoorbeeld in de duiding illustreert de opzet: we wensen *précondities* aan formele parameters op te leggen door middel van attributen. In het hieronder uitgewerkte voorbeeld zal een *préconditie* enkel van toepassing zijn op een individuele parameter. We wensen de volgende situtatie te bekomen:

- bij de start van een methode zal het *OnEntry* aspect de eventueel aanwezige parameter-attributen (te vinden in de formele methode declaratie) checken;
- elk préconditie attribuut verifieert of de actuele parameter waarde voldoet. Indien niet wordt er een fout gegooid;
  - we leggen aan elk préconditie attribuut de interface *IAOPValidatie* op. Op deze manier is het gemakkelijk na te gaan of een parameter attribuut door onze infrastructuur moet gechecked worden;
  - we voorzien die interface van een *valideer* methode. Op die manier kan elk attribuut op dezelfde manier verwerkt worden (zodat we met een eenvoudige lus alle attributen kunnen overlopen en verwerken).

Meer in detail gaan we voor de attributen als volgt te werk

- definieer de interface *IAOPValideer* en voorzie de *valideer*- methode. Het return type is *void* en de routine heeft één parameter van type *object* (op deze manier kan het valideren van een datum of string door dezelfde virtuele methode opgevangen worden;
- definieer préconditie klasses die deze interfaces implementeren. De constructor (gebruikt in de attribuut definitie van de formele parameter) neemt welilcht één of meerdere parameters (range- condities nemen wellicht twee argumenten in de constructor);
- implementeer de *valideer*- methode. Deze throwt een *AOPValidatieException* indien de voorwaarde niet voldaan is.

Voor het aspectorientatie gedeelte voorzien we volgende code:

- voorzie een (*PostSharp*) *OnMethodBoundaryAspect*;
- override de *OnEntry*- methode. Via *args.Method.GetParameters* bekomt u de formele parameters van de oproep (*args* bevat de actuele parameters). De formele en actuele parameters stemmen positioneel met elkaar overeen (de eerste actuele parameter (postie 0) stemt overeen met de eerste formele parameter (ook positie 0)).
- schrijf wat code om het voldaan zijn van de eventuele attributen te checken (twee lussen, vijf lijntjes logica code (dus zonder accolades of routine hoofdingen))!

Houd ook rekening met volgende zaken:

- wanneer u uw code test, beperk u dan niet tot précondities die niet voldaan zijn. Indien de préconditie wel voldaan is moet het programma uiteraard verder werken;
- plaats eens meerdere précondities op hetzelfde argument;
- in de *PostSharp* documentatie vinden we dat het construeren van de method- informatie een inefficient proces is. De hierboven geschetste implementatie zal deze informatie bij elke methode oproep verzamelen. U kan dit beperken door de methode informatie te verzamelen in de *RuntimeInitialize* uitvoering, en deze dan op module niveau (voor deze specifieke methode, dus niet *statisch*) te bewaren.

Hopelijk realiseert u zich dat de attributen die worden geplaatst bij de formele parameter definities op zich geen AOP introduceren: we plaatsen met deze attributen meta-informatie op de parameters, die dan door de AOP-verwerking (de method- entry) gebruikt worden om de extra logica (de aspect logica) te sturen.

lol, dat werkt!!!

14.2.3 PostSharp en performance counters

Performance Counters zijn (sinds NT) *de* manier om prestatie cijfers omtrent uw toepassing aan het operating systeem mede te delen. Third party toepassingen kunnen deze counters lezen en visualiseren (of rapporteren):

Figuur 14.3: PDFStamper performance counter

Bovenstaande figuur toont de performancemonitor resultaten van mijn PDFWatermark toepassing (op elke bladzijde wordt een watermerk geplaatst). U merkt duidelijk dat er twee uitvoeringen van het programma voorkwamen. De hier getoonde performance monitor zit standaard in Windows (de screenshot is genomen in een Vista- installatie).

Windows en .NET leveren reeds een grote lijst performance counters. U kan deze gemakkelijk nakijken vanuit de *Server Explorer* in *Visual Studio*:

Figuur 14.4: Performance counters in Visual Studio

Bij een eerste test zal u wellicht geïnteresseerd zijn in hardware gerelateerde counters zoals de *batterijstatus* of *processor* gegevens.

lol, dat werkt!!

14.2.3.1 Eigen performance counters

Voor het publiceren van eigen performance counters verwijs ik naar How do I... Publish a custom application performance counter in $C^{\#}$?. U zal dit uiteraard moeten beheersen vooraleer u uw eigen performance counters via AOP kan aansturen (het PerformanceCounterType is zeer belangrijk).

14.2.3.2 AOP en performance counters

De performance counters worden vooral gebruikt om toepassingen te profileren: hoe gedraagt een toepassing zich tijdens haar uitvoering? Performance counters kunnen dan ook best gebruikt worden om betekenisvolle activiteiten van uw toepassing te publiceren. Denken we hierbij bijvoorbeeld aan: het aantal vewerkte klanten (per seconde), het aantal failures (per seconde), .. . Om dit met AOP te ondersteunen denken we aan volgende implementatie:

- we wensen een performance attribuut op methode niveau toe te passen, en de mogelijkheid te hebben om op *OnEntry*, *OnException* en *OnSuccess* in te grijpen. *OnMethodBoundaryAspect* is hiervoor uitermate geschikt;
- op basis van de gegevens, nodig om een goede performance counter te definiëren, zullen we bij het overerven extra parameters aan het attribuut toevoegen;
- gebruik deze extra info om op de gewenste plaatsen een performance counter aan te sturen: u bent natuurlijk *niet* verplicht om zowel het falen als slagen van een methode uitvoering te registreren.

14.3 AOP oefeningen

- definieer een attribuut, plaatsbaar op een formele parameter, dat gecombineerd met AOP (ook zelf te maken) verifieert dat een integer waarde zich tussen een onder- en bovengrens bevindt;
- definieer een attribuut, plaatsbaar op een formele parameter, dat gecombineerd met AOP (ook zelf te maken) verifieert dat een geboortedatum een reeds geboren persoon (datum niet in de toekomst) van maximaal 120 jaar definieert;
- definieer een AOP-attribuut, plaatsbaar op een methode, waarmee wordt afgedwongen dat de uitvoerder van een routine een administrator op het toestel moet zijn;
- definieer een performance counter die met behulp van AOP op methodes kan worden geplaatst. De counter telt het aantal uitvoeringen van een routine. Gebruik deze om het aantal uitvoeringen van een aantal van de recursieve methode oproepen uit de voorbije lessen te tellen:
  - tel het aantal recursieve oproepen in uw fibonacci- oefening;
  - tel het aantal recursieve oproepen in uw backtrack- oefeningen (N-Queens en Sudoku)
  - tel het aantal PDF pagina's dat geschreven wordt met uw PDF- merge oefening.

  Het resultaat van deze performance counters moet in de performance monitor zichtbaar zijn

lol, dat werkt!!!

lol, dat werkt!!!
Jeroen Gaels

Hoofdstuk 15

Query Notification

Query Notification biedt de mogelijkheid om gegevens wijzigingen in de database via een event in uw client programa op te vangen. Deze techniek vooral zinvol wanneer u weinig veranderende gegevens cacht in uw client (het bijhouden van lijsten informatie).

Het opzetten van *Query Notifications* is niet zo moeilijk maar vergt enig lees- en opzet werk.

15.1 Query notification oefening

Demonstreer het kunnen werken met Query Notification.

lol, dat werkt!!!

Jef Daels
lol, dat werkt!!!

Hoofdstuk 16

Discussie topics

16.1 Duiding

Als ontwikkelaar moet u in team kunnen werken. In dit team zijn wellicht ook andere ontwikkelaars aanwezig. De hieronder beschreven discussie topics worden door de klas besproken tot een consensus omtrent oplossing wordt bekomen:

- er wordt slechts één algoritme per klas per topic aanvaard;
- iedereen implementeert dit algoritme tegen de volgende les (absentie systeem);
- de week erna volgt een nabespreking, die wellicht resulteert in een herdiscussie;
- de begeleidende docent probeert de discussie *niet* te sturen.

Naast de oplossing voor de technische problemen zijn vooral volgende elementen van belang:

- ook al bent u overtuigd van het werken van uw oplossing, toch moet u ook de oplossing van iemand anders kunnen aanhoren:
  - welke punten uit zijn oplossing zijn een verbetering voor mijn eigen oplossing?
  - welke punten uit zijn oplossing zijn minder goed dan mijn eigen oplossing, en hoe kan ik dit beargumenteren
- in een werksituatie is het niet genoeg om gelijk te hebben: u moet de collega's overtuigen van uw gelijk (vermijd het Cassandra syndroom)
  - hoe krijg ik een complexe redenering uitgelegd? gebruik ik voorbeelden? terminologie?
  - hoe beschrijf ik het foutieve in een redenering?

lol, dat werkt!!!

16.2 Sinus berekenen mbv Taylor reeks

Het berekenen van een sinus kan gebeuren door middel van een Taylor reeks. U vindt een beschrijving hiervan op Wikipedia (Taylor reeks).

Groepsdiscussie: bespreek de te implementeren oplossing.

lol, dat werkt!!!
Jeroaels

Hoofdstuk 17

Global Assembly Cache: GAC

17.1 Global Assembly Cache: duiding

.NET herintroduceerde *copy deployment*, reeds bekend vanuit het *DOS*-tijdperk. Copy deployment bestaat erin dat de installatie directory van de ene naar de andere computer wordt gecopieerd, waarna dit programma ook op de tweede computer werkt.

In een tijd toen disk-space duur was (het DOS tijdperk) introduceerde dit veel overtollige copiën van dezelfde, door iedereen gebruikte dll's. Ook is het niet altijd gemakkelijk voor een programma om het reeds geïnstalleerd zijn van een ander programma te achterhalen: in theorie moet je de ganse schijf doorzoeken. Om aan deze nadelen tegemoet te komen ontwikkelde Microsoft de *registry*: elk programma dat zich installeerde registreerde zichzelf en zijn onderdelen in de registry. Nieuwe versies werden over bestaande versies heen gecopieerd (hun locatie was te vinden in de registry), en programma's vonden elkaar terug door de registry te ondervragen.

lol, dat werkt!!!

Dit gebruik van de registry resulteerde in tal van problemen, in detail beschreven in DLL hell(Wikipedia):

- indien een nieuwe versie niet 100% backwards compatibel is zal een programma dat vroeger werkte dit potentieel niet meer doen: het is perfect mogelijk om twee programma's te ontwikkelen die niet op hetzelfde toestel geïnstalleerd kunnen worden;
- het installeren van twee versies van hetzelfde programma is quasi onmogelijk;

.NET herintroduceert copy-deployment: elke toepassing komt met zijn eigen set dll's in zijn installatie directory: er is geen versie conflict meer. Voor systeem routines is dit een moeilijk houdbare zaak: sommige dll's zullen we toch centraal voor alle toepassingen wensen te installeren. Dit gebeurt in de Global Assembly Cache.

17.1.1 Relevante namespaces

17.2 GAC: technische duiding

pputer ▸ Local Disk (C:) ▸ Windows ▸ assembly ▸

Assembly Name	Version	Cul...	Public Key Token	Proces...
Expression.DevHost	3.0.0.40...		31bf3856ad364e35	x86
Expression.DevHost.resources	3.0.0.40...	en	31bf3856ad364e35	x86
Extensibility	7.0.330...		b03f5f7f11d50a3a	
GAC2008	8.0.0.0		dcc8364b2810e1dc	MSIL
GAC2008	7.0.0.0		dcc8364b2810e1dc	MSIL
GAC2008	5.0.0.0		dcc8364b2810e1dc	MSIL
GAC2008	1.0.0.0		dcc8364b2810e1dc	MSIL
IACore	1.7.622...		31bf3856ad364e35	
IALoader	1.7.622...		31bf3856ad364e35	
IIEExecRemote	2.0.0.0		b03f5f7f11d50a3a	MSIL
IIEHost	2.0.0.0		b03f5f7f11d50a3a	MSIL
IIEHost	2.0.0.0		b03f5f7f11d50a3a	MSIL
Interop.Dso	5.2.0.0		89845dcd8080cc91	MSIL

Figuur 17.1: Global Assembly Cache (iets oudere versie, Win 7)

Microsoft.Web.Administration.resources	7.9.0.0	es	31bf3856ad364e...	MSIL
Microsoft.Web.Administration.resources	7.9.0.0	de	31bf3856ad364e...	MSIL
Microsoft.Web.Administration.resources	7.9.0.0	cs	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation	7.1.0.0		31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	zh-CHT	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	zh-CHS	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	tr	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	ru	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	pt	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	pl	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	ko	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	ja	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	it	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	fr	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	es	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	en	31bf3856ad364e...	MSIL
Microsoft.Web.Delegation.resources	7.1.0.0	de	31bf3856ad364e...	MSIL

Figuur 17.2: Global Assembly Cache (iets oudere versie, Win 8)

- er komen assemblies met identieke namen voor (dit is onmogelijk voor files in gewone directories). Dit maakt het mogelijk om in de GAC aan versie beheer te doen (dit voorkomt de problemen die optreden wanneer incompatibele versies elkaar overschrijven);
- de assemblies hebben een versie nummer. Hierdoor wordt het mogelijk om een programma te koppelen aan een specifieke versie van een assembly;

Niet zichtbaar op de screenshots, maar toch relevant:

- een assembly in de GAC *moet strong named* zijn;
- bij installatie van een nieuwe versie van een assembly kan met een *policy*- file opgegeven worden welke versies vervangen worden: het *redirecten* van assemblies. Deze *policy* files worden ook in de *GAC* geplaatst, en moet dan ook strongly named zijn. Op deze manier kan een upgrade automatisch geactiveerd worden voor programma's die een verouderde assembly gebruiken;
- een programma config file kan dit redirecten overschrijven, en verwijzen naar nog een andere (of de oorspronkelijke) assembly. De config file van het programma heeft voorrang op de assembly policy file.

  Het is belangrijk dat de programma config file voorrang heeft op de assembly policy file: indien een nieuwe versie onverhoopt toch problemen oplevert, kan men per individueel programma beslissen met welke assembly versie men blijft verder werken.
- vanaf Visual Studio 2010 is de GAC meer folder georiënteerd:
  C:\Windows\Microsoft.NET\assembly\GAC_MSIL\GACDLL\v4.0_1.0.0.0__72e3a55536524714

Figuur 17.3: Eigen GAC-dll's, Win 8)

17.3 Ontwikkelen voor en met de GAC

17.3.1 Strong named assemblies

Het MSDN artikel omtrent Strong-named assemblies beschrijft duidelijk *het waarom* omtrent named assemblies:

Strong-Named Assemblies

msdn

A strong name consists of the assembly's identity — its simple text name, version number, and culture information (if provided) — plus a public key and a digital signature. It is generated from an assembly file (the file that contains the assembly manifest, which in turn contains the names and hashes of all the files that make up the assembly), using the corresponding private key. Microsoft® Visual Studio® .NET and other development tools provided in the .NET Framework SDK can assign strong names to an assembly. Assemblies with the same strong name are expected to be identical.

You can ensure that a name is globally unique by signing an assembly with a strong name. In particular, strong names satisfy the following requirements:

- Strong names guarantee name uniqueness by relying on unique key pairs. No one can generate the same assembly name that you can, because an assembly generated with one private key has a different name than an assembly generated with another private key.
- Strong names protect the version lineage of an assembly. A strong name can ensure that no one can produce a subsequent version of your assembly. Users can be sure that a version of the assembly they are loading comes from the same publisher that created the version the application was built with.
- Strong names provide a strong integrity check. Passing the .NET Framework security checks guarantees that the contents of the assembly have not been changed since it was built. Note, however, that strong names in and of themselves do not imply a level of trust like that provided, for example, by a digital signature and supporting certificate.

When you reference a strong-named assembly, you expect to get certain benefits, such as versioning and naming protection. If the strong-named assembly then references an assembly with a simple name, which does not have these benefits, you lose the benefits you would derive from using a strong-named assembly and revert to DLL conflicts. Therefore, strong-named assemblies can only reference other strong-named assemblies.

See Also

Global Assembly Cache | Signing an Assembly with a Strong Name

Figuur 17.4: Strong-named assemblies (MSDN)

Om zelf een strong-named assembly te creeëren gaan we als volgt te werk:

- maak een nieuw project aan (voor dit voorbeeld een class library, hoewel natuurlijk ook alle andere types strong-named kunnen zijn);
- via de project properties in Visual Studio, in de signing farde kiest u:
  - *sign the assembly*;
  - kies een bestaande *strong name key file*, of maak een nieuwe aan. Hierdoor wordt aan het project een *snk*- file toegevoegd. Wanneer u het project herbuild zal de resultaat assembly strong-named zijn.
    - via *sn.exe*- commando kan u ook zelf een key-pair file aanmaken (zie ook How to: Create a Public/Private Key Pair.

- omdat we deze DLL later zullen gebruiken om assembly redirecting te testen zullen we een klasse voorzien die het versie nummer van de assembly teruggeeft.

  System.Reflection.Assembly.GetExecutingAssembly().GetName().Version is een startpunt om de major, minor en build gegevens van een assembly op te vragen.

17.3.2 Gacutil.exe

Dit commando laat toe om de GAC te manipuleren en ondervragen. MSDN beschrijft de parameterlijst, even googlen brengt u op de internetsnelweg.

Indien u een niet *strong-named* dll in de GAC probeert te plaatsen krijgt u het volgende resultaat:

Figuur 17.5: Gacutil -i Strong-named failure

Wanneer de assembly wel strong-named is:

Figuur 17.6: Gacutil -i Strong-named success

U kan de aanwezigheid in de cache verifëren:

Figuur 17.7: Gacutil -l

17.3.3 DLL-versies maken

Een assembly versie bestaat uit major, minor, build en revision getallen. U kan deze instellen via de project properties (application, assembly information):

Figuur 17.8: Assembly versies

U kan de versie van een file opvragen via zijn properties:

Figuur 17.9: File properties

Het is perfect mogelijk om ook deze DLL in de GAC te plaatsen. Op dat moment bevat de GAC twee dll's van dezelfde assembly.

Jeff Daels lol, dat

17.3.4 Redirecting assemblies

17.3.4.1 Welke assembly wordt geladen?

Het zoeken naar assemblies wordt beschreven in How the Runtime Locates Assemblies. Hier vindt u onder andere:

The runtime uses the following steps to resolve an assembly reference:

1. Determines the correct assembly version by examining applicable configuration files, including the application configuration file, publisher policy file, and machine configuration file. If the configuration file is located on a remote machine, the runtime must locate and download the application configuration file first.
2. Checks whether the assembly name has been bound to before and, if so, uses the previously loaded assembly.
3. Checks the global assembly cache. If the assembly is found there, the runtime uses this assembly.
4. Probes for the assembly using the following steps:
   a. If configuration and publisher policy do not affect the original reference and if the bind request was created using the Assembly.LoadFrom method, the runtime checks for location hints.
   b. If a codebase is found in the configuration files, the runtime checks only this location. If this probe fails, the runtime determines that the binding request failed and no other probing occurs.
   c. Probes for the assembly using the heuristics described in the probing section. If the assembly is not found after probing, the runtime requests the Windows Installer to provide the assembly. This acts as an install-on-demand feature.

Note There is no version checking for assemblies without strong names, nor does the runtime check in the global assembly cache for assemblies without strong names.

Figuur 17.10: Assembly Loading

Belangrijk hierin op te merken zijn volgende elementen:

- welke versie van toepassing is is afhankelijk van een aantal configuratie of policy files;
- indien geen configuratie of policy redirections gedefinieerd zijn is de at compile time gebruikte versie van toepassing, ook al is er een nieuwere versie in de GAC te vinden;
- indien de assembly al eerder werd geladen, dan wordt deze gebruikt;
- de GAC heeft voorrang op alle andere locaties. Uiteraard enkel van toepassing voor strong-named assemblies, omdat dit een voorwaarde is om in de GAC geplaatst te kunnen worden.

Eventuele uitvoeringsproblemen bij het laden van een assembly kan u documenteren met behulp van fuslogvw.exe.

17.3.4.2 Redirecting met configuration file

Een volledige write-up omtrent assembly redirection vindt u in Redirecting Assembly Versions. Onderstaande tekst is slechts een beperkte weergave van de vermeldde MSDN-entry. Het hier uitgewerkte voorbeeld maakt gebruik van de twee DLL-versies die we in de GAC plaatsten: versie 1.0.0.0 en 2.1.0.0.

- maak een WPF applicatie die refereert naar de recenste versie en gebruik daarin de klasse met het assembly versie nummer om het de assembly versie te tonen;
  - deze DLL wordt trouwens niet naar de *bin* directory gecopieerd;
- voer de toepassing uit en check de assembly versie;
- indien we de toepassing wensen te redirecten naar een andere versie (1.0.0.0) zullen we in de configuratie file wat extra XML voorzien:

The following example shows how to redirect one version of `myAssembly` to another, and turn off publisher policy for `mySecon`

```
<configuration>
   <runtime>
      <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
       <dependentAssembly>
         <assemblyIdentity name="myAssembly"
                           publicKeyToken="32ab4ba45e0a69a1"
                           culture="en-us" />
         <!-- Assembly versions can be redirected in application, publisher policy, or machine configura
         <bindingRedirect oldVersion="1.0.0.0"
                          newVersion="2.0.0.0"/>
       </dependentAssembly>
       <dependentAssembly>
         <assemblyIdentity name="mySecondAssembly"
                           publicKeyToken="32ab4ba45e0a69a1"
                           culture="en-us" />
          <!-- Publisher policy can be set only in the application configuration file. -->
          <publisherPolicy apply="no">
       </dependentAssembly>
      </assemblyBinding>
   </runtime>
</configuration>
```

Figuur 17.11: configredirection

  - het *public key token* kan u op volgende wijzes bekomen:
    * gebruik *gacutil -l*: de opgelijste informatie bevat het token;
    * zoek in de assembly cache folder (als onderdeel van diretory naam);
    * voeg een external tool toe in Visual Studio, zoals beschreven in Visual Studio Tip: Get Public Key Token for a Strong Named Assembly. Het voordeel van deze manier van werken is het copy/paste beschikbaar zijn van het token binnenin Visual Studio.

- de *culture* tage is optioneel. Indien afwezig (of *neutral*) is de redirect van toepassing op alle cultures (taal);
- u kan ook ranges opgeven bij het redirecten.

17.3.4.3 Redirecting met policy files

De aangeraden writeup: How to: Create a Publisher Policy Ga op volgende wijze te werk:

- wijzig de config file in de bin-directory van de WPF-applicatie zodat er geen redirection meer toegepast wordt: de gebruikte dll-versie is 2.1.0.0. Check dit door de de gecompileerde versie uit te voeren;
- wijzig de DLL-versie naar 3.0.0.0 en hercompileer de DLL (*niet* de applicatie). De toepassing werkt nog steeds met versie 2.1.0.0.
- maak een *policy.config* file in de directory waar de keypair file staat. Plaats hierin de XML-code om 2.1 te redirecten naar 3.0.
- gebruik de assembly linker *al.exe* zoals beschreven in How to: Create a Publisher Policy om een policy file aan te maken;
  - ikzelf navigeer de commandprompt met *cd* commando's naar de directory waarin zich de *snk* en *policy* file bevinden. Op deze manier vermijd ik lange filenamen (en bijhorende tikfouten) bij het intikken;
  - merkwaardig genoeg zal in de policy-file de *major.minor* van de te vervangen dll worden opgenomen, en *niet* deze van de vervangende dll die nu geïnstalleerd wordt als vervanger;
- plaats de policy file in de GAC. Om hem terug te vinden moet u in de GAC zoeken op de prefix *policy*;

lol, dat werkt!!!

```
Administrator: Visual Studio Command Prompt (2010)

D:\prevdata\PIH\vakken\20102011\programmer solutions\lessen\DemoGAC\GACDLL>al /link:policy
.config /out:policy.2.1.GACDLL.dll /keyfile:gacdll.snk /platform:anycpu
Microsoft (R) Assembly Linker version 10.0.30319.1
Copyright (C) Microsoft Corporation. All rights reserved.


D:\prevdata\PIH\vakken\20102011\programmer solutions\lessen\DemoGAC\GACDLL>gacutil /i poli
cy.2.1.GACDLL.dll
Microsoft (R) .NET Global Assembly Cache Utility.  Version 4.0.30319.1
Copyright (c) Microsoft Corporation.  All rights reserved.

Assembly successfully added to the cache

D:\prevdata\PIH\vakken\20102011\programmer solutions\lessen\DemoGAC\GACDLL>
```

Figuur 17.12: Policy file commando's

- plaats de nieuwe DLL versie in de GAC;

- test uw toepassing: deze moet nu versie 3.0 gebruiken;
- wijzig de config file terug zodat versie 1.0 gebruikt wordt. Dit illustreert de voorrang van de configuration file op de policy file.

17.3.4.4 Redirecting op machine niveau

Dit gebeurt opnieuw met configuration files en maakt geen onderdeel uit van deze cursus.

17.3.4.5 Redirecting redirections

Dit lijkt niet mogelijk te zijn:

- maak een programma dat gebruik maakt van versie 1.0 van een DLL;
- maak een policy file die versie 1.0 vervangt door versie 2.0;
- maak een configuration file die versie 2.0 vervangt door versie 3.0;
- plaats alle versies in de GAC;
- bij uitvoering wordt versie 2.0 gebruikt: herhaald redirecting lijkt dus niet te werken (wat ook overeenstemt met het artikel dat beschrijft hoe een assembly gevonden wordt)

lol, dat werkt!!!

17.3.5 GAC en project references

Voor een volledige toelichting: How to: Add or Remove References in Visual Studio. Belangrijke is het volgende element:

- GAC assemblies komen *niet* voor in de references dialogen

Dll's gevonden in de GAC worden ook niet opgenomen in de bin-directory van de client-toepassing.

17.3.6 Installeren in de GAC vanuit de Setup (tem VS 2010)

In een Visual Studio installatie project kan u een *special folder* opnemen die verwijst naar de GAC:

Figuur 17.13: GAC en setup

Indien u hier (strong named) assemblies opneemt zullen deze in de GAC van het target toestel geïnstalleerd worden.

17.4 GAC opgave

Implementeer de situatie die hierboven wordt beschreven en hieronder summier herhaald wordt:

- maak een WPF toepassing die gebruik maakt van een eigen DLL. Deze DLL biedt onder andere de mogelijkheid om zijn assembly versie te ondervragen. De applicatie gebruikt dit om de gebruikte DLL versie te tonen;
- alle versies van de DLL komen in de GAC terecht;
- gebruik een config file om te redirecten, u kiest tussen welke versies;
- gebruik een policy file om te redirecten;
- toon aan dat de configuration file redirection voorrang heeft op de policy file redirection.

17.5 Extras

addin om vanuit visual studio assembly redirection te implementeren manifest?

lol, dat werkt!!!
Jeroaels

Hoofdstuk 18

Visual Studio geautomatiseerde broncode manipulaties

18.1 Visual Studio geautomatiseerde broncode manipulaties: duiding

Een moderne IDE helpt de ontwikkelaar bij het produceren van code: zowel het reduceren van tikwerk (de IDE genereert bijvoorbeeld de proxy klasses gebruikt in een WCF-oplossing) als het vermijden van foutieve code (intellisense) zijn hiervan voorbeelden. In dit hoofdstuk komen volgende elementen aan bod: toolbox codeblocks, snippets, macro's, add-ins en (kort) het gebruik van MEF in Visual Studio 2010.

lol, dat werkt!!!

18.2 Relevante namespaces

- EnvDTE: een namespace die informatie omtrent Visual Studio projecten bevat (*alle* types projecten);
  - *Changes and new functionality are contained in the EnvDTE80, EnvDTE90, EnvDTE90a, and EnvDTE100 namespaces.*: er zijn dus upgrades
- VslangProj100: in deze assembly wordt extra informatie omtrent programmeer projecten bijgehouden die men niet kwijt kan in *EnvDTE* (bijvoorbeeld: referenties)
  - *The VslangProj100 namespace defines types used to automate project tasks. These types extend the types defined in the VSLangProj, VSLangProj2, VSLangProj80, and VslangProj90 namespaces.*: ook hier zijn dus verschillende (opeenvolgende) versie (upgrades) beschikbaar.

18.3 Toolbox codeblocks

Wanneer we in het code-window vertoeven is de toolbox meestal leeg. Het is mogelijk om een blok code in de editor te selecteren en deze op de toolbox te slepen, waarna we (via right- click) dit stukje code ook een naam kunnen geven (optioneel). Eens op de toolbox geplaatst kan u dit stukje code overal copiëren door te klikken op zijn toolbox- voorkomen:

Figuur 18.1: Code in de toolbox

18.3.1 Voordeel van toolboxcode

- zeer gemakkelijk op te zetten en te gebruiken: alle manipulaties blijven binnen Visual Studio;

18.3.2 Nadelen van toolboxcode

- de benadering is weinig flexibel: er wordt een letterijke copy aangemaakt die geen enkele ondersteuning biedt bij aanpassingen;
- deze manier van werken is niet vlot porteerbaar tussen toestellen.

18.4 Code snippets

Snippets bieden de mogelijkheid om template code in een source-file te pasten. In de resultaat code is het mogelijk om intelligente vervangingen door te voeren (wijzigen op één plaats van een stukje code zal copy code, elders in het snippet ook wijzigen). Visual Studio wordt aangeleverd met tal van snippets out-of-the box en biedt de mogelijkheid om ook eigen snippets toe te voegen.

18.4.1 Snippets gebruiken

Een snippet wordt geïdentificeerd door een *shortcut* (bijvoorbeeld *propa* voor een attached dependency property). Indien u deze shortcut intikt, gevolgd door twee *TAB* toetsaanslagen wordt de gekoppelde snippet-code in uw bronfile gecopieerd:

```
#region "adres"
prop
```

- DependencyPropertyHelper
- DependencyPropertyKey
- FrameworkPropertyMetadata
- FrameworkPropertyMetadataOptions
- prop
- propa
- propdp
- Properties

Define an attached DependencyProperty
Code snippet for an attached property using DependencyProperty as

Figuur 18.2: propa snippet gebruiken

Bemerk ook het icoontje dat Visual Studio aan een snippet koppelt. Wanneer we met deze snippet code genereren krijgen we het onderstaande resultaat:

```
public static int GetMyProperty(DependencyObject obj)
    {
    return (int)obj.GetValue(MyPropertyProperty);
    }

public static void SetMyProperty(DependencyObject obj, int value)
    {
    obj.SetValue(MyPropertyProperty, value);
    }

// Using a DependencyProperty as the backing store for MyProperty.  This enables animation, styling, bindin
public static readonly DependencyProperty MyPropertyProperty =
    DependencyProperty.RegisterAttached("MyProperty", typeof(int), typeof(ownerclass), new UIPropertyMetada
```

Figuur 18.3: propa snippet resultaat

De tekst met achtergrondkleur zal, indien hij wordt gewijzigd, ook andere tekst aanpassen

(zeker het proberen waard!). Bij het zelf aanmaken van snippets zullen we aangeven hoe deze verschillende codeelementen aan elkaar gekoppeld worden.

18.4.2 Eigen code snippets

Het *propa*-snippet genereert de dependency property registratie en de wrapper methodes. Ik zelf heb bij het definiëren van *attached dependency properties* veelal ook nood aan de propertyChanged callback, waarvan ik de signatuur niet van buiten ken: ik wou dat ik een *snippet* had die deze extra code bevatte. Het is mogelijk om de bestaande snippet aan te passen (mijns insziens minder wenselijk) of een nieuwe snippet aan te maken (een snippet is een gewone ascii-file). We zullen een eigen snippet aanmaken: *propachanged*.

U vindt de geïnstalleerde snippets in subdirectories van *Program Files\Microsoft Visual Studio 10.0\VC#\Snippets\1033*. Het *propa-* snippet wordt gevonden in de *NetFX30* subdirectory. We zullen dit snippet copiëren naar een eigen directory waarna we het snippet zullen aanpassen:

- copieer *propa.snippet* naar een eigen snippet folder;
- pas de beschikbare snippet informatie aan;
- pas de code aan en voorzie een change routine die gekoppeld wordt via de metadata, zodat de property-naam onderdeel uitmaakt van de change-routine, en mee gewijzigd wordt indien u de property naam wijzigt.

Opdat een snippet in Visual Studio gekend zou zijn moet u deze bibliotheek importeren:

- *Tools.Code Snippets Manager*;
- kies de taal waarvoor uw snippet werd ontwikkeld, en voeg uw snippet directory toe.

Hierna kan u uw zopas aangemaakt snippet testen.

lol, dat werkt!!!

18.4.3 Voordelen van snippets

- deze manier van werken kan gedeeld worden door verschillende ontwikkelaars (u vind op het internet vele voorbeelden);
- de snippet-code ondersteunt een vorm van pseudo parametristatie (en is dus iets dynamischer in gebruik dan de toolbox code);

18.4.4 Nadeel van snippets

- echte parametrisatie is nog niet mogelijk: het is bijvoorbeeld onmogelijk om een snippet te maken die in zijn textexpansie het huidige tijdstip opneemt;

Indien u de cursustekst niet leuk vond kan u ook op Extending Visual Studio Part 1 - Creating Code Snippets terecht.

18.5 Macro's in Visual Studio

Net zoals Office (Visual Basic for Applications) biedt Visual Studio de mogelijkheid om gebruikershandelingen te registereren zodat ze op een later tijdstip opnieuw uitgevoerd kunnen worden: *macro*'s. Omdat deze commando's als programma code worden bewaard kunnen ze ook aangepast worden om ook in niet identieke situaties bruikbaar te zijn. Ikzelf gebruik macro's vooral om voorbeeld code te genereren die ik dan (aangepast) overneem bij het ontwikkelen van Add-ins. Op die manier kan het onderzoeken van het Visual Studio objectmodel sterk beperkt worden (net zoals in Office worden macro's geregistreerd met Visual Basic).

18.5.1 Macro samples

Vanonder *Tools.Macros.Macro Explorer* krijgen we toegang tot de beschikbare macro's. De samples bevatten (bruikbare) voorbeeld macro's die richtinggevend zijn voor de mogelijkheden. U kan ze uitvoeren (double click) of editeren (via right click). Belangrijk hierbij is dat ze ook debugbaar zijn: u kan breakpoints plaatsen om de logica van de macro na te volgen:

- onder *DevStudio6Editor* vindt u de macro *CommentOut*. Deze zal een highlighted stuk code in commentaar plaatsen. Probeer dit in uw programma, normaal gezien zal dit niet werken. Gebruik de debugger mogelijkheden om na te pluizen waarom dit niet werkt, en pas dit aan.

lol, dat werkt!!!

18.5.2 Eigen macro's

In deze les zullen we zelf geen eigen macro's ontwikkelen maar onmiddellijk overstappen naar het ontwikkelen van addIn's.

18.5.3 Voordelen van macro's

- programmeer kennis is niet nodig om een macro op te nemen (*Tools.Macros* helpt u verder);
- een macro kan een zeer gemakkelijke manier zijn om startcode te bekomen voor het maken van een Add-In (we moeten het objectmodel van Visual Studio niet zelf uitpluizen).

18.5.4 Nadeel van macro's

- het macro recorden beperkt zich tot het registreren door middel van VB-code (voor een nMCT-student niet echt onoverkomelijk);
- macro's bieden weinig mogelijkheden tot visuele feedback. Dit is voor ons de reden om geen eigen Macro te ontwikkelen en onmiddellijk over te stappen naar Add-ins.

18.6 Visual Studio Addin's

Een Addin nestelt zich in de toolbar of het menu van uw Visual Studio omgeving en manipuleert uw Visual Studio solution. Omdat het foutief manipuleren van Visual Studio grote problemen kan veroorzaken (Visual Studio onbruikbaar kan maken) wordt in een test-fase gewerkt met een Experimental Build. Indien er iets fout loopt gebeurt dit in deze copy omgeving, en blijft uw originele Visual Studio bruikbaar.

- om addins te ontwikkelen installeert u Microsoft Visual Studio SDK;
- u kan de experimental hive (de registry entries gebruikt door de experimental build) resetten door *Visual Studio 2010 SDK.tools.Reset the Microsoft Visual Studio Experimental Hive* te kiezen.

18.6.1 Creatie van een Visual Studio Add-in

Om een Addin aan te maken gaan we als volgt te werk:

- maak een nieuw project aan met als projecttype *Other Project Types.Extensibility.Visual Studio Add-in*;
  - bij het doorlopen van de wizard kiezen we als programmeertaal *VB.NET*. Bij het programmeren van de Add-In's worden veel properties als *Object* opgeleverd. In die situaties is het handig om gebruik te maken van de late binding mogelijkheden van VB.NET;
    - u kan in *C#*een gelijkaardig effect als *late binding* bekomen door gebruik te maken van het keyword dynamic. Ikzelf vind het gemakkelijker om gewoon in VB.NET te programmeren:
      - automatische late binding;
      - de macro gegenereerde code is VBA-code, en daardoor meer lijkend op VB.NET dan op *C#*.
  - application host is de toepassing waarbinnen de add-in bruikbaar moet zijn;
  - bij het invullen van 'Choose Add-in Options' is de commentaar bij elke checkbox volgens mij duidelijk genoeg om zelfstandig de keuzes te maken;

lol, dat werkt!!!

18.6.2 Testuivoering van een Add-in

Indien u het project nu opstart wordt een nieuwe Visual Studio gestart waarin deze Addin actief is. Fouten (door de nieuwe Add-in) in deze omgeving hebben geen invloed op de originele Visual Studio omgeving.

Indien de add-in niet geladen wordt in de extra gestartte omgeving kan het eventueel helpen om in de startup options van het project volgende commandline parameter mee te geven:

- */ranu /rootsuffix Exp*

De aangemaakte Add-In bevat standaardaard al flink wat code. We zullen ons concentreren op de *Connect* file:

- de constructor van de Connect- klasse laat toe om extra initialisatie code uit te voeren;
- *OnConnectionConnect!OnConnection* wordt uitgevoerd wanneer de Add-In gekoppeld wordt aan Visual Studio. Normaal gezien zullen we deze code ongewijzigd laten;
- de methode *Exec* wordt uitgevoerd wanneer de Visual Studio gebruiker de Add-In start, en bevat de eigenlijke logica van de Add-In.

18.6.3 Een lijnenteller Add-In

We zullen de add-in aanpassen zodat we in een listview per project in de solution alle files en hun aantal lijnen tonen. Met deze Add-In krijgen we een idee over de grootte van een solution.

- maak een formulier dat deze informatie zal tonen en start dit formulier vanuit de *Exec* methode (van de Connect- klasse van de Add-In);
  - momenteel is het enkel mogelijk om *WinForms* Add-in formulieren aan te maken;
  - in de ctor geeft u wellicht best het applicationobject van de addin mee (een object dat de applicatie, in dit geval Visual Studio bevat);
- *EnvDTE* is de namespace waarin u de hier nodige klasses (intefaces) terugvindt. In combinatie met de intellisense en wat zoeken kan u de lijnenteller hopelijk opzetten;
  - projectitems kunnen genest zijn: zo zijn designer generated code files projectitems van de origineel toegevoegde files;
  - niet elk projectitem heeft een overeenkomstige file;

lol, dat werkt!!!

Wees kritisch omtrent uw output: indien er maar één file gevonden wordt in uw project is er zeer veel kans dat uw Add-In foutief werkt!

18.6.4 Reference historiek Add-In

EnvDTE bevat enkel die informatie die gemeenschappelijk is aan alle Visual Studio projecten (setup, VB.NET, website, silverlight, phone, ..). Extra programmeer specifieke informatie vindt men in *VsLangProj*. Instanties van *EnvDTE* types kunnen gecast worden naar hun *VsLangProj* tegenhanger via de *.object* property van het *EnvDTE* object.

Als oefening zullen we een Add-in maken die voor elk project in een solution een file aanmaakt (datum staat in de filenaam) waarin de gebruikte referenties worden weggeschreven. Normaal gezien zou ik dit in een database registereren, maar op deze manier zijn we ook verplicht een extra file aan een project toe te voegen (een goede oefening). Met dergelijke informatie is het mogelijk te achterhalen welke projecten potentieel probleem hebben bij het verlopen of upgraden van een dll.

- maak terug een Visual Basic .NET Add-In;
- maak een macro voor de volgende acties:
  - selecteer een project in een solution;
  - voeg aan dit project een nieuwe file toe;
  - tik iets in deze nieuwe file
- in de Add-In ontwikkelen we code die per project een reference historiek file aanmaakt en daarin de references wegschrijft. Gebruik de macrocode als startcode om dit te realiseren.
  - de macro code werkt als macro code. In een Add-In werkt de code, om een nieuwe file toe te voegen niet. Gelukkig wordt in ItemOperations.AddNewItem Method duidelijk beschreven hoe de argumenten voor het toevoegen moeten geschreven worden;
  - uw code moet gebruik maken van een solution name en project name (enige nazicht in de macro code maakt duidelijk waarom). Dit komt niet automatisch, u zal hier wat code moeten voor schrijven;
- indien u de nieuw aangemaakt file wenst te saven kan u in de macro best ook een *save all* instructie opnemen.

lol,dat werkt!!!

18.7 Visual Studio events

Gedurende een Visual Studio sessie doen zich vele events voor waar een Add-In kan op inpikken: het toevoegen van items, het openen en of sluiten van projecten zijn hiervan voorbeelden. Indien u hierop wenst in te pikken kan u dit op volgende wijze:

```
Private solEvents As SolutionEvents
Private tekstEvents As TextEditorEvents
Private Sub startupComplete()
    solEvents = _applicationObject.Events.SolutionEvents
    AddHandler solEvents.ProjectAdded, AddressOf projAdded
    AddHandler solEvents.Opened, AddressOf solOpened

    tekstEvents = _applicationObject.Events.TextEditorEvents
    AddHandler tekstEvents.LineChanged, AddressOf LineChanged
End Sub
Private Sub solOpened()
    MessageBox.Show("sol opened")
End Sub
Private Sub projAdded(ByVal proj As EnvDTE.Project)
    MessageBox.Show("proj added")

End Sub
Private Sub LineChanged(ByVal start As EnvDTE.TextPoint, ByVal eind As Env
    MessageBox.Show("item added")
End Sub
```

Figuur 18.4: Eventhandling in een Add-In

lol.

Belangrijk in bovenstaande code zijn volgende elementen:

- de events- variabelenw waarvoor events worden opgevangen moeten op moduleniveau geplaatst worden (zie ook Getting DocumentEvents after solution is opened).
- het koppelen van events gebeurt in VB.NET niet door middel van de +=- operator maar met behulp van een *AddHandler*- instructie. Uiteraard moet ook hier de signatuur van de gekoppelde routine geschikt zijn;
- het grootste probleem is wellicht het vinden van het gepaste event block. In het voorbeeld komen volgende zaken aan bod:
  - het openen van een solution;
  - het toevoegen van een project
  - het wijzigen van een lijn code

18.8 Deployment van een Add-In

Indien een Add-In naar behoren werkt wensen we deze niet langer in de experimental hive uit te voeren: we wensen de Add-In te gebruiken in een normale Visual Studio omgeving, of de add-In te verspreiden naar andere computers. Om dit te realiseren gaan we als volgt te werk:

- lees How to create and load add-ins in Visual Studio .NET by using Visual $C^{\#}$.NET or in Visual Studio 2005 by using Visual $C^{\#}$2005 en voer uit wat daarin beschreven staat (er ontbreken belangrijke stappen, maar een goed begrijper met voorkennis kan die wel oplossen)
  - pas deze manier van werken toe voor de references Add-In;
  - bij mij werkt het niet;
- de ouderwetse manier (zonder .msi file) die enkel werkt indien u Visual Studio installeerde (omdat we de referenties niet meeinstalleren):
  - via *Tools.Options* achterhalen we de directories waarin Add-Ins gezocht worden:

Figuur 18.5: Addin directories

  - in mijn installatie bestond de *Addins*- directory niet: ik heb deze manueel aangemaakt

  - copieer de *.addin* file die Visual Studio in uw bronfile (bij de code files) plaatste, tesamen met de *.dll* uit de compilatie directory naar de Add-In directory die u koos (indien u niet één van de aanwezige directories koos moet u uw eigen directory toevoegen)
  - bij het heropstarten zal visual studio de AddIn terugvinden en integereren;

- u kan het al dan niet activeren van Add-Ins aansturen vanonder *Tools.Add-in Manager*.

18.9 Visual Studio en MEF: extentions

Zoals in het hoofdstuk MEF werd aangehaald is Visual Studio 2010 MEF-gebaseerd. De technologie op zich is relatief gemakkelijk te begrijpen, maar het voorzien van extenties in een complexe software als Visual Studio is toch niet evident. Extending the Editor is een potentieel startpunt om u hierin te verdiepen. In het vervolg van deze tekst zullen we de intellisense uitbreiden om een gemakkelijk commentaarregels aan code toe te voegen. Uiteindelijk wordt ook toegelicht hoe deze extenties kunnen *gedeployed* worden.

Hoewel we nu enkel zullen ingrijpen op de intellisense kan er natuurlijk veel meer gerealiseerd worden. Een aantal voorbeelden (met broncode) hiervan kan u terugvinden op MSDN Developer Samples: Learn Through Code waarbij u met behulp van het zoekvenster zoekt op *VSX*

lol, dat werkt!!!

18.9.1 cmnt extention

Visual Studio gebruikt intellisense om de ontwikkelaar bij het coderen mogelijke aanvullingen te presenteren. Dit verhoogt de tiksnelheid, vermijdt fouten en vereist gememoriseerde namen van de ontwikkelaar. Het is de bedoeling om via intellisense een commentaarregel te kunnen toevoegen waarin zowel de naam van de ontwikkelaar als het tijdstip van toevoegen voorkomt:

Figuur 18.6: Cmt intellisense

- dit kan niet met snippets gerealiseerd worden omdat deze uit vaste tekst bestaan: de auteur en het tijdstip zouden vaste waardes zijn die achteraf gewijzigd moeten worden.

18.9.2 cmt extention implementatie

- Walkthrough: Displaying Statement Completion diende als leidraad om mijn eigen *cmnt* completion aan te maken: volg de volledige beschrijving;

- het voorbeeld past intellisense toe op *PlainText* files en werkt dus enkel op txt-files, en niet in *C#*of VB- code files. Om dit te verhelpen moet u het *ContentType* aanpassen (op twee! plaatsen);
  - natuurlijk moet u bij het overnemen van de code de gepaste wijzigingen aanbrengen: u moet de *completions* van het voorbeeld aanpassen. Wie de code begrijpt zal daar weinig problemen bij ondervinden;
  - u zal moeten opzoeken wel ContentType u dan wel dient te gebruiken.
- ik heb niet de pretentie die code voldoende te begrijpen om ze zelf van scratch te ontwikkelen.

18.9.3 cmt deployment

Indien u een extention wenst te installeren is copy deployment onvoldoende. Er is nood aan een *VSIX*- file, die bij dubbelklikken de extentie installeert. Het aanmaken van een VSIX-file wordt beschreven in Packaging an Extension in an Empty VSIX Project. Beter dan die telst kan ik de topic niet beschrijven.

18.9.4 Extentions en Visual Studio

Visual Studio heeft een *Tools.Extention Manager*. Hiermee kunnen geïnstalleerde extentions gedisabled of uninstalled worden.

Ik dat werkt!!

18.10 Relevante literatuur

- Creating and Using IntelliSense Code Snippets: alles wat u ooit over snippets wou weten, en nog veel meer!
- Getting Started with Extending Visual Studio
- Extending the Editor
- VSX Home on Code Gallery

18.11 Codemanipulatie opgaves

- ontwikkel een snippet waarmee een attached dependency property wordt aangemaakt, aangevuld met een change- handler. De change handler routine naam is gekoppeld aan de property naam (en wordt dus gewijzigd indien de property naam wordt gewijzigd);
- ontwikkel een Add-In om voor elke file in elk project van een solution het aantal codelijnen te tellen;
- ontwikkel een Add-In om voor elk project in een solution de referenties in een file bij te houden. Elke uitvoering maakt een eigen versie van deze file. U neemt de datum en het tijdstiop hiervoor op in de filenaam;
- (optioneel) maak een extention die toelaat om een commentaar lijn (zoals hierboven beschreven) toe te voegen aan een code window.

lol, dat werkt!!!

Jef Daels
lol, dat werkt!!!

Figuren

2.1 Extention method definition . . . 13
2.2 Extention method gebruik . . . 13
2.3 Extention method gebruik: fadeto . . . 14
2.4 Await demo . . . 15
2.5 Await demo async methode . . . 15

3.1 De klasse nMCTDocent . . . 17
3.2 DisplayMemberPath binding . . . 18
3.3 BindingPathError . . . 18
3.4 ILSpy DisplayMemberPath info . . . 19
3.5 ViewModel klasse . . . 20
3.6 Application Resources . . . 20
3.7 Window binding . . . 20
3.8 DataContext binding . . . 21
3.9 ItemsSource declaratie . . . 21
3.10 Observable hoofding . . . 22
3.11 Collection hoofding . . . 22
3.12 ItemsSource binding . . . 22
3.13 ItemTemplate instellen . . . 22
3.14 nMCTDocent datatemplate . . . 23
3.15 nMCTDocenten in een ComboBox . . . 23
3.16 RelativeSource binding . . . 24

4.1 Delegate definitie . . . 26
4.2 Delegate demo: uppercase en lowercase . . . 26
4.3 Event definitie (zonder guidelines) . . . 28
4.4 Event abonnering . . . 28
4.5 Event afvuren (zonder guidelines) . . . 29
4.6 Static event definitie (volgens de guidelines) . . . 31
4.7 Static event client . . . 31
4.8 Asynchrone verwerking demo . . . 34
4.9 AsyncCallback demo . . . 35
4.10 UI-thread problem . . . 35
4.11 Dispatcher demo . . . 36
4.12 Await- keyword . . . 37

lol, dat werkt!!!

4.13 XMLViewer en code plumbing . . . 40
4.14 Anonymous method voorbeeld . . . 42
4.15 Lambda expressie voorbeeld . . . 42

5.1 ComponentModel attributen ($C^{\#}$) . . . 46
5.2 Assembly reflection ($C^{\#}$) . . . 47
5.3 Strongly typed instantiatie . . . 48
5.4 Activator.CreateInstance ($C^{\#}$) . . . 48
5.5 Activator.InvokeMember ($C^{\#}$) . . . 49

6.1 Commandline compilatie: $C^{\#}$broncode . . . 54
6.2 $C^{\#}$commandline compilatie en uitvoering . . . 55
6.3 Commandline compilatie: VB.NET broncode . . . 55
6.4 VB.NET commandline compilatie voorbeeld . . . 56
6.5 $C^{\#}$System.CodeDom.Compiler namespace . . . 57
6.6 CodeDOM . . . 58
6.7 ILDasm output . . . 59
6.8 ILSpy output . . . 60
6.9 Reflector output . . . 61
6.10 Obfuscated output via Reflector . . . 62
6.11 Tekenen van een functie . . . 63

7.1 NQueens oplossen . . . 66
7.2 Sudoku spelregels: een opgave . . . 67
7.3 Sudoku spelregels: een oplossing . . . 68
7.4 Sudoku routed events demo . . . 70
7.5 Awaiting een oproep . . . 71
7.6 TaskCompletionSource: SetResult . . . 71
7.7 TaskCompletionSource demo . . . 72
7.8 4 op een rij: bordwaarde . . . 73
7.9 Minimax algoritme . . . 75

8.1 Attached dependency property in XAML . . . 80
8.2 Attached dependency property definitie ($C^{\#}$) . . . 81
8.3 F1 infrastructuur ($C^{\#}$) . . . 82
8.4 CLR binding error . . . 84
8.5 Dependency property syntax 1 ($C^{\#}$) . . . 84
8.6 Dependency property syntax 2 ($C^{\#}$) . . . 85
8.7 Breedte animatie ($C^{\#}$) . . . 85
8.8 Background animatie . . . 86
8.9 Transformatie van een Button ($C^{\#}$) . . . 87
8.10 Transformatie animatie ($C^{\#}$) . . . 88
8.11 ValueConverter definitie ($C^{\#}$) . . . 89
8.12 ValueConverter XAML ($C^{\#}$) . . . 90
8.13 XAML binding dependency property . . . 92

lol, dat werkt!!!

9.1 Window met eigen caption . . . 94
9.2 Window met WindowStyle none . . . 95
9.3 Checkbox states visualisatie . . . 98
9.4 Checkbox states via Blend . . . 98
9.5 State animaties in XAML . . . 99
9.6 State en Parte opgave: verkeerslicht . . . 99

10.1 Height en applicationsettings . . . 105
10.2 Enumeration definitie . . . 107

11.1 Sudoku in PDF . . . 110

12.1 Kwadraat code . . . 114
12.2 Kwadraten selecteren . . . 115

14.1 AOP: parameter annotatie . . . 123
14.2 PostSharp en ILSpy . . . 125
14.3 PDFStamper performance counter . . . 127
14.4 Performance counters in Visual Studio . . . 128

17.1 Global Assembly Cache (iets oudere versie, Win 7) . . . 136
17.2 Global Assembly Cache (iets oudere versie, Win 8) . . . 136
17.3 Eigen GAC-dll's, Win 8) . . . 137
17.4 Strong-named assemblies (MSDN) . . . 138
17.5 Gacutil -i Strong-named failure . . . 139
17.6 Gacutil -i Strong-named success . . . 139
17.7 Gacutil -l . . . 139
17.8 Assembly versies . . . 140
17.9 File properties . . . 140
17.10 Assembly Loading . . . 141
17.11 configredirection . . . 142
17.12 Policy file commando's . . . 143
17.13 GAC en setup . . . 145

18.1 Code in de toolbox . . . 148
18.2 propa snippet gebruiken . . . 149
18.3 propa snippet resultaat . . . 149
18.4 Eventhandling in een Add-In . . . 155
18.5 Addin directories . . . 156
18.6 Cmt intellisense . . . 157

lol, dat werkt!!!

Index

Action, 41
Activator, 48
    CreateInstance, 48
Add-ins, 152
AddHandler, 96, 155
addOwner, 83
AddressOf, 29
AffectsMeasure, 91
AffectsRender, 91
al.exe, 143
Alfa Beta pruning, 75
Animaties, 85
Animation
    ColorAnimation, 86
    DoubleAnimation, 85
    Transformation, 87
Anonymous method, 42
AOP, 123
    ContextBoundObject, 123
Application settings, 105
Assembly, 45
Async, 37
async, 15
AsyncCallback, 34
Asynchrone verwerking, 34
    UI-thread, 36
Attach to proces, 116
Attached dependency property, 79
Attribute
    Category, 46
    DefaultEvent, 46
    DefaultProperty, 46
    DefaultValue, 46
    Description, 46
Await, 37
await, 15
backtracking, 65
BeginInvoke, 34
Behavior, 103
    Drag, 102
    Grip, 103, 105
Behaviors, 101
BesteZet, 74
Blend, 88
Bootstrapper Project, 121
BordWaarde, 73
Brush, 86
bubbling, 32

CanConvertFrom, 90
Category, 46
ccCaption, 95
Closed, 106
cmnt token, 157
Code snippet, 149
CodeDomProvider, 57
Color, 86
ColorAnimation, 86
CompilerParameters, 57
CompilerResults, 58
Connect, 153
    Connect, 153
    Exec, 153
ContextBoundObject, 123
contravariance, 27
Convert, 89
ConvertBack, 89
ConvertFrom, 90
covariance, 27
CreateInstance, 48
CreateProvider, 57

DataContext, 19
DefaultEvent, 46

lol, dat werkt!!!

DefaultProperty, 46
DefaultValue, 46
Delegate, 25, 45
    Asynchrone verwerking, 34
    BeginInvoke, 34
    codeplumbing, 39
    events, 27
    multithreading, 33
Dependency properties, 79
DependencyProperty, 81
    addOwner, 83
    RegisterAttached, 81
Description, 46
deserialisatie, 89
Dispatcher, 36
DisplayMember, 18
DisplayMemberPath, 18
DoubleAnimation, 85

EAP, 37
EnvDTE, 147, 153
Event based Asynchronous Pattern, 37
EventArgs, 30
Events
    +=, 28
    AddHandler, 29
    AddressOf, 29
    event keyword, 28
    EventArgs, 30
    guidelines, 30
    handles, 28
    onEvent, 30
    preview, 32
    RaiseEvent, 29
    routed events, 32
    static events, 31
    tunneling event, 32
Events:bubbling event, 32

F1URL, 80
Func, 41

GAC, 135
Gacutil.exe, 139
Generics, 30
GetCustomAttributes, 49
GetLanguageFromExtension, 57
GetTemplateChild, 96
Ghost data, 33
Global Assembly Cache, 135
Grip, 103

IAOPValidatie, 125
IAsyncResult, 34
IL, 58, 59
ILDasm, 59
inconsistent read, 33
InstallShield, 119
instructies
    lock, 33
intermediate language, 59
InvokeMember, 49
ItemsControl, 21, 22
ItemsSource, 21
ItemsTemplate, 22
iTextSharp, 109
IValueConverter, 89, 90
    Convert, 89
    ConvertBack, 89

Lambda expression, 42
LinearGradientBrush, 89
LINQ, 42
Lost update, 33

Macros, 151
Main, 54
MEF, 157
MenuItem, 52
MiniMax, 73
    BesteZet, 74
    BordWaarde, 73
    MiniMax, 75
    Pruning, 75
MouseDownEvent, 102
MouseDragElementBehavior, 102
MouseMoveEvent, 102
MSI, 119
Multithreading, 33

N-Queen, 66

lol, dat werkt!!!

Namespace
System, 25
System.CodeDom.Compiler, 53
System.ComponentModel, 45, 46, 79
System.Reflection, 45
System.Windows, 79
System.Windows.Data, 79
System.Windows.Media, 79
nMCTDocent, 17

Obfuscation, 62
ObservableCollection, 17
OnApplyTemplate, 94, 96
OnEntry, 124, 125
OnException, 125
OnMethodBoundaryAspect, 124
OnRender, 91
OpenFileDialog, 47

PARTS, 94
patterns
strategy pattern, 39
Performance counter, 127
PerformanceCounterType, 128
plugin, 45
policy, 137
PostSharp, 123, 124
OnEntry, 124
OnException, 125
OnMethodBoundaryAspect, 124
preview, 32
PreviewKeyDownEvent, 69
Process, 82
propa, 149
propachanged, 150
PropertyPath, 86
Pruning, 75

Query Notification, 131

redirect, 137
Reflection, 45
Reflector, 61
registry, 135
Resize, 104
runtime compilatie, 53
RuntimeInitialize, 126

Server Explorer, 128
signatuur, 27
sn.exe, 138
Snippet, 149
propa, 149
snippet
propa, 81
propdp, 81
snk, 138
SQL Server, 113
SQL Server Datatools, 113
States, 97
StoryBoard, 86
TargetNameProperty, 86
TargetPropertyProperty, 86
String, 54
Strong name, 138
Sudoku, 110
System, 25, 54
System.CodeDom.Compiler, 53, 57
System.ComponentModel, 45, 46, 79
System.Reflection, 45
System.Windows, 79
System.Windows.Data, 79
System.Windows.Media, 79

Table Value Functions, 115
Task, 71
TemplatePart, 94
Themes, 96
Thread
Sleep, 38
Toolbox code, 148
Transformation, 87
TryParse, 90
tunneling, 32
Type.GetCustomAttributes, 49
Type.InvokeMember, 49
TypeConverter, 89, 90

UI-thread, 36
using, 54

ValueConversion, 89

lol,dat werkt!!

Version, 139
Visual Studio, 128
VisualStateManager, 99
VisualTreeHelper, 96, 103
    GetParent, 96
VSIX, 158
VslangProj100, 147

Window, 50
WindowStyle, 95
WiX, 119

XMLViewer, 39

lol, dat werkt!!!