Prism

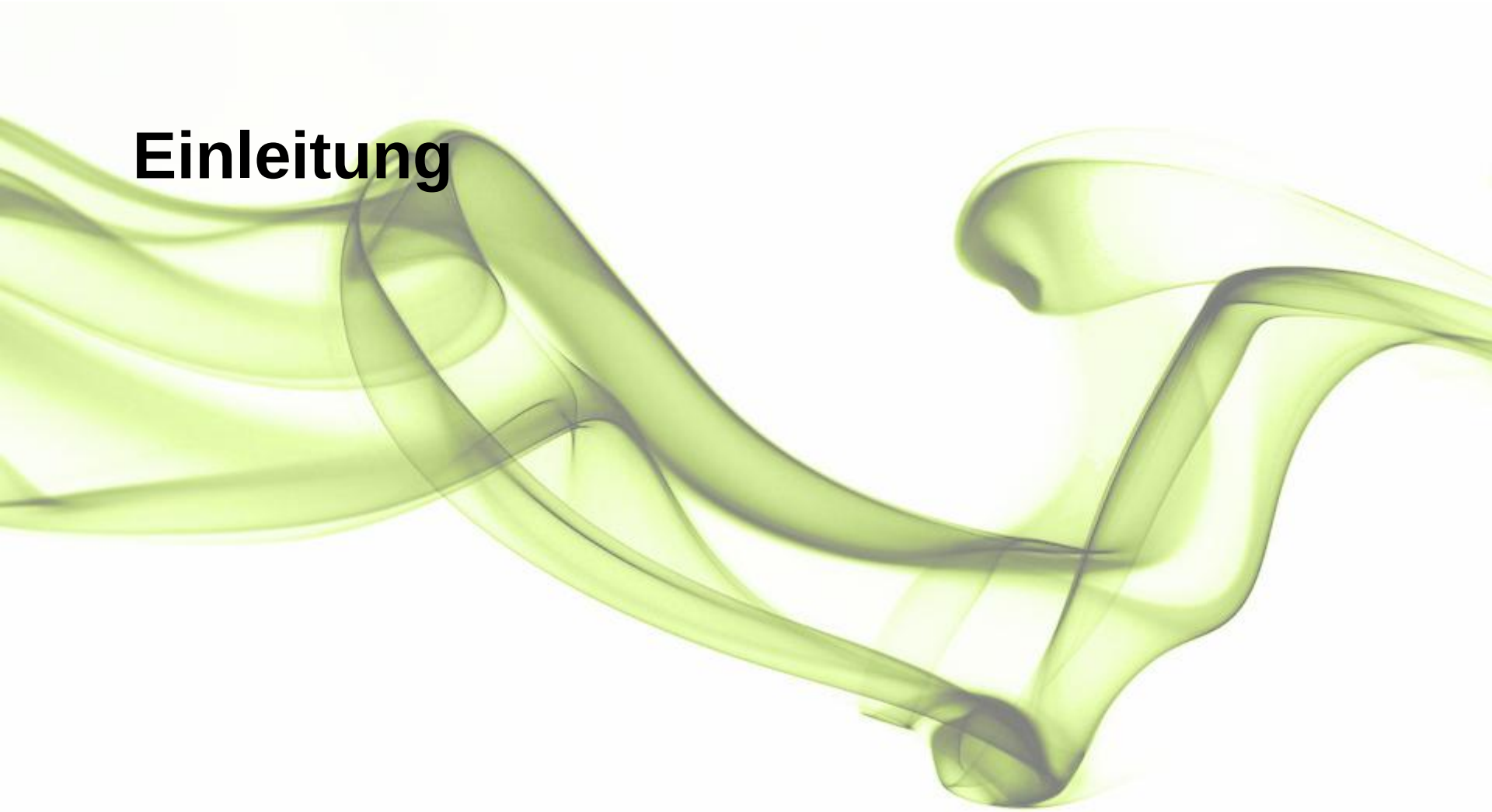
Martin Hey
Unique Software e.K.

Zielsetzung

Dieser Workshop liefert einen Überblick darüber:

- was Prism ist und wo die Vorteile liegen
- wie man Anwendungen unter der Verwendung von Prism aufbaut
- was man unter Modulen, Regions und Views versteht und wie diese zusammenspielen
- wie Kommunikation zwischen den Komponenten einer Prism-Anwendung funktioniert
- welche Navigationsmechanismen es gibt

Einleitung



Überblick

- Framework zur Modularisierung von Anwendungen
- Trennung von Aufgaben und Zuständigkeiten
- für WPF, Silverlight und Windows Phone 7
- verwendet dazu verschiedene Designpattern, z.B.
  - MVVM
  - Inversion of Control
  - Dependency Injection
  - Command-Pattern

Key-Konzepte

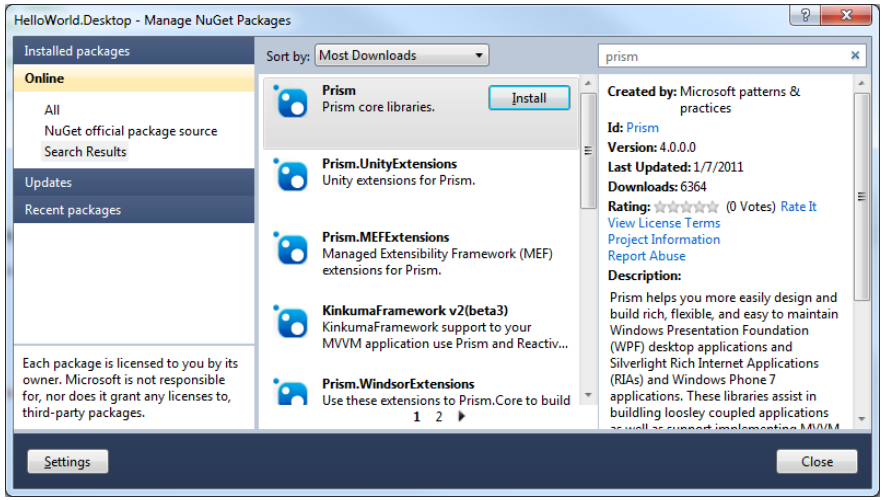
- Shell
  - Template für die Benutzeroberfläche
- Regions
  - Flächen innerhalb der Shell in welche Views zur Laufzeit geladen werden können
- Module
  - Funktionsblock der Anwendung
  - beinhalten Views
- Views
  - Benutzerinteraktion
- Bootstrapper
  - Initialisierung der Prism-Anwendung

Einbinden in die eigene Solution

- Download

  http://compositewpf.codeplex.com

- NuGet-Package



Bootstrapper und Shell

Bootstrapper

- Initialisiert die Anwendung (Services)
- Core-Services von Prism
  - IModuleManager
  - IModuleCatalog
  - IModuleInitialize
  - IRegionManager
  - IEventAggregator
  - ILoggerFacade
  - IServiceLocator
- Anwendungsspezifische Services die von allen Modulen verwendet werden sollen

Shell

- Main Window / Main Page
- Template für die Anwendung
- besteht aus Regions in die zur Laufzeit Views integriert werden

Beispiel: Bootstrapper und Shell mit Unity

- WPF-Anwendung erstellen und Referenzen auf Prism und die PrismUnityExtensions setzen

- App.xaml und App.xaml.cs anpassen

```
<Application x:Class="WpfApplication1.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             >
    <Application.Resources>

    </Application.Resources>
</Application>
```

```
protected override void OnStartup(StartupEventArgs e)
{
    base.OnStartup(e);
    var bootstrapper = new Bootstrapper();
    bootstrapper.Run();
}
```

Beispiel: Bootstrapper und Shell mit Unity

- Bootstrapper erstellen

```
public class Bootstrapper : UnityBootstrapper
{
    protected override DependencyObject CreateShell()
    {
        throw new NotImplementedException();
    }
}
```

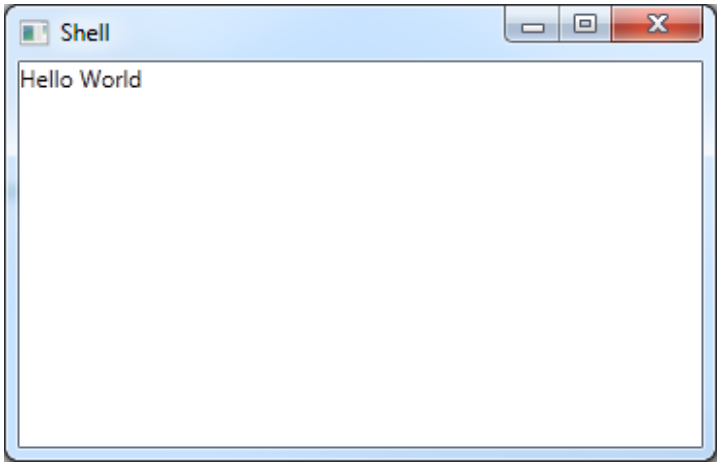
- Shell erstellen

```
<Window x:Class="WpfApplication1.Shell"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Shell">
    <Grid>
        <TextBlock Text="Hello World" />
    </Grid>
</Window>
```

Beispiel: Bootstrapper und Shell mit Unity

- Bootstrapper anpassen

```
using System.Windows;
using Microsoft.Practices.Prism.UnityExtensions;
using Microsoft.Practices.Unity;

namespace WpfApplication1
{
    public class Bootstrapper : UnityBootstrapper
    {
        protected override DependencyObject CreateShell()
        {
            return Container.Resolve<Shell>();
        }

        protected override void InitializeShell()
        {
            base.InitializeShell();
            App.Current.MainWindow = (Window) Shell;
            App.Current.MainWindow.Show();
        }
    }
}
```

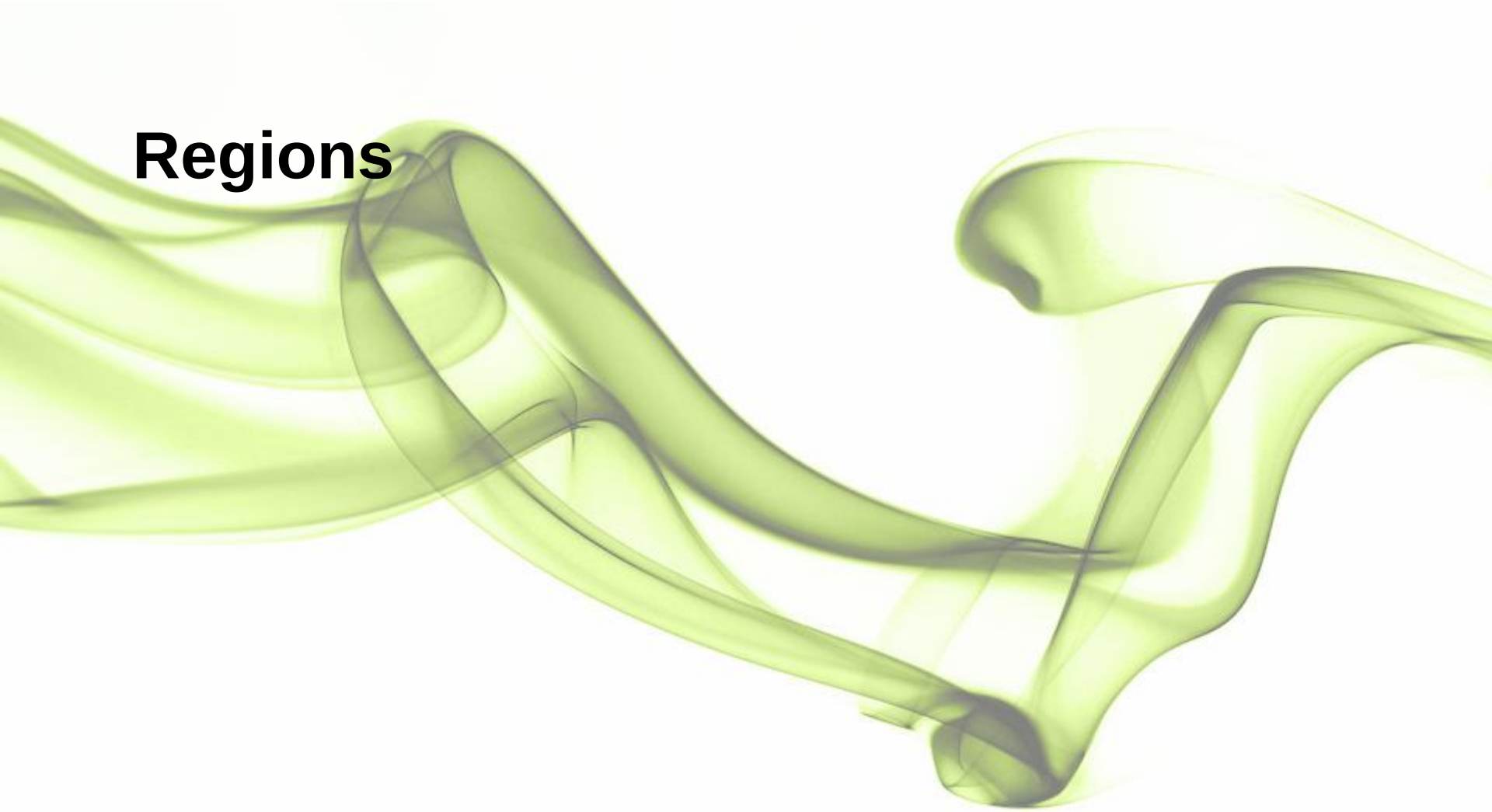


ServiceLocator

- Wrapper um den DI-Container, um die Dependencies aufzulösen
  => verwendeter DI-Container könnte später ausgetauscht werden

```
protected override DependencyObject CreateShell()
{
    return ServiceLocator.Current.GetInstance<Shell>();
}
```

Regions



Regions

- benannter Platzhalter für dynamisch geladene Inhalte in der Shell
- z.B. Toolbar-Region und Content-Region
- kennen keine Views
- Layout der Shell kann angepasst werden, die Inhalte werden an die korrekten Stellen injected
- implementieren IRegion

RegionManager und RegionAdapter

- RegionManager
  - verwaltet die Regions der Anwendung
  - stellt die Attached Property RegionName bereit
  - mapt RegionAdapter und Controls
  - stellt die Attached Property RegionContext bereit

- RegionAdapter
  - ContentControlRegionAdapter
  - ItemsControlRegionAdapter
  - SelectorRegionAdapter
  - TabControlRegionAdapter (nur Silverlight)

Beispiel: Ein Modul laden

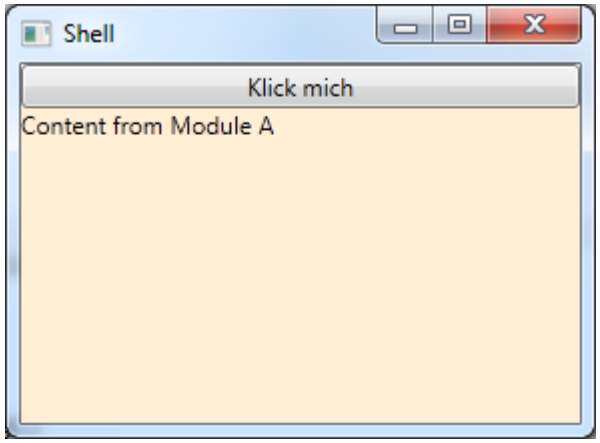
- Module als Bibliothek anlegen
  - ToolbarView-Control (UserControl mit Button)
  - ContentView-Control (UserControl mit Inhalt)
  - Module-Klasse (implementiert IModule)

```
public class ModuleAModule : IModule
{
    private readonly IUnityContainer _container;
    private readonly IRegionManager _regionManager;

    public ModuleAModule(IUnityContainer container, IRegionManager regionManager)
    {
        _container = container;
        _regionManager = regionManager;
    }

    public void Initialize()
    {
        _regionManager.RegisterViewWithRegion("ToolbarRegion", typeof(ToolbarView));
        _regionManager.RegisterViewWithRegion("ContentRegion", typeof(ContentView));
    }
}
```

Beispiel: Ein Modul laden

- Bootstrapper anpassen

```
protected override void ConfigureModuleCatalog()
{
    base.ConfigureModuleCatalog();
    ModuleCatalog.AddModule(new ModuleInfo("ModuleA", typeof(ModuleAModule).AssemblyQualifiedName));
}
```



Eigene Regions implementieren

- wenn ein 3rd-Party-Control eine Region sein soll oder ein Control, das diese Möglichkeit nicht standardmäßig mitbringt
- Schritte:
  - ableiten von RegionAdapterBase<T>
  - implementieren der CreateRegion-Methode
    - SingleActiveRegion (ContentControls)
    - AllActiveRegion (ItemsControls)
    - Region (Selector)
  - implementieren der Adapt-Methode
  - registrieren des Adapters im Bootstrapper

Beispiel: StackPanel als Region-Host

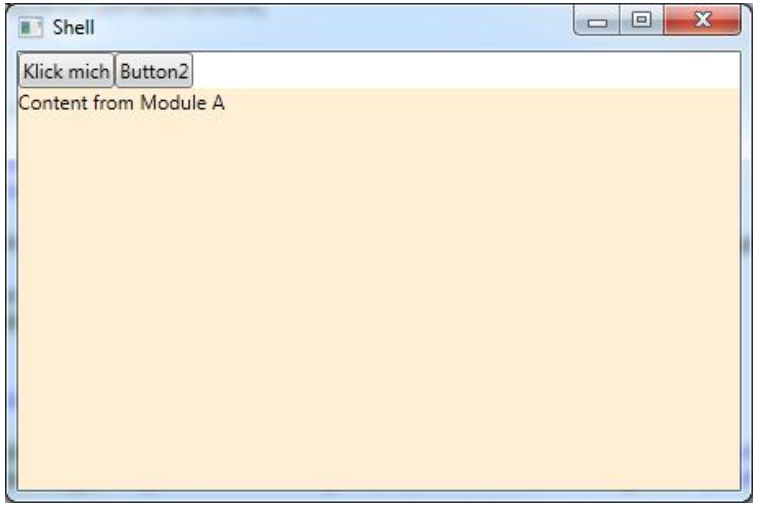
- implementieren des RegionAdapters

```
public class StackPanelRegionAdapter : RegionAdapterBase<StackPanel>
{
    public StackPanelRegionAdapter(IRegionBehaviorFactory regionBehaviorFactory) : base(regionBehaviorFactory)
    {
    }

    protected override void Adapt(IRegion region, StackPanel regionTarget)
    {
        region.Views.CollectionChanged += (s, e) =>
                                          {
                                              // handle add
                                              if (e.Action == NotifyCollectionChangedAction.Add)
                                              {
                                                  foreach (FrameworkElement element in e.NewItems)
                                                  {
                                                      regionTarget.Children.Add(element);
                                                  }
                                              }

                                              // handle remove
                                          };

    }

    protected override IRegion CreateRegion()
    {
        return new AllActiveRegion();
    }
}
```

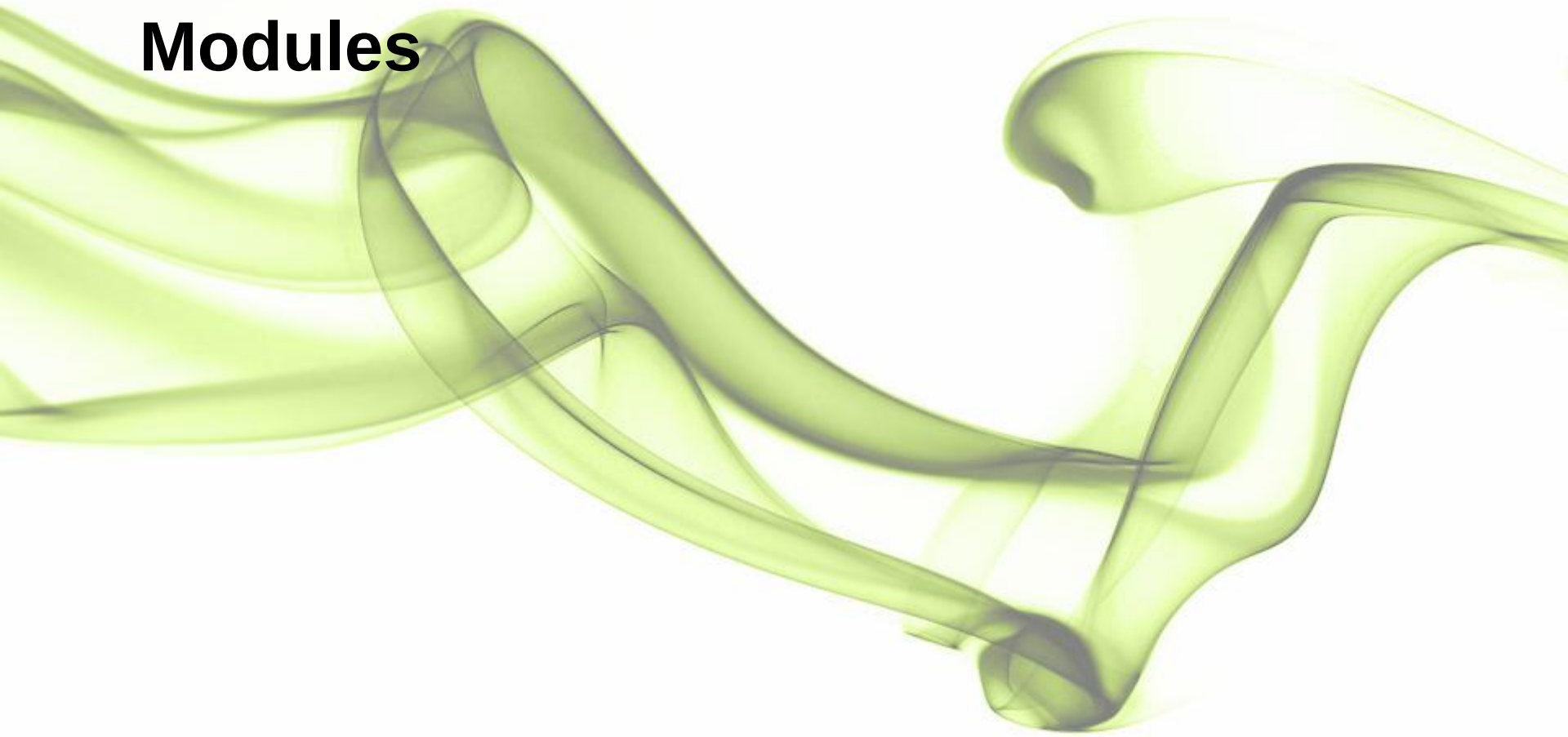
Beispiel: StackPanel als Region-Host

- registrieren im Bootstrapper

```
protected override Microsoft.Practices.Prism.Regions.RegionAdapterMappings ConfigureRegionAdapterMappings()
{
    var mappings = base.ConfigureRegionAdapterMappings();
    mappings.RegisterMapping(typeof(StackPanel), Container.Resolve<StackPanelRegionAdapter>());
    return mappings;
}
```



Modules



Modules

- Baustein der Anwendung
- Paket, dass Funktionalität und Ressourcen eines Teils der Anwendung kapselt
- beinhaltet eine zentrale Klasse, die IModule implementiert

Modules

- aufgenommen im ModuleCatalog per
  - Code
  - XAML
  - Konfigurationsdatei
  - Ordner im Dateisystem
- Ladeverhalten anpassbar
  - sobald verfügbar
  - sobald benötigt (on demand)
- Initialize-Methode
  - Registrierung von Typen und Services
  - Integrieren von Views in die Shell
  - Abonnieren von Services oder Events

Modules registrieren

- per Code (Nachteil: Assembly muss referenziert sein)

```
protected override void ConfigureModuleCatalog()
{
    var moduleAType = typeof (ModuleAModule);
    var moduleAInfo = new ModuleInfo
                          {
                              ModuleName = moduleAType.Name,
                              ModuleType = moduleAType.AssemblyQualifiedName,
                              InitializationMode = InitializationMode.WhenAvailable
                          };
    ModuleCatalog.AddModule(moduleAInfo);
}
```

- aus Ordner

```
protected override IModuleCatalog CreateModuleCatalog()
{
    return new DirectoryModuleCatalog
               {
                   ModulePath = @".\Modules"
               };
}
```

```
[Module(ModuleName = "ModuleA", OnDemand = true)]
public class ModuleAModule : IModule
```

Modules registrieren

- aus Ressourcendatei

```
<Modularity:ModuleCatalog xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                          xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                          xmlns:Modularity="clr-namespace:Microsoft.Practices.Prism.Modularity;assembly=Microsoft.Practices.Prism">

    <Modularity:ModuleInfo Ref="file://ModuleA.dll"
                           ModuleName="ModuleA"
                           ModuleType="ModuleA.ModuleAModule, ModuleA, Version=1.0.0.0"
                           InitializationMode="WhenAvailable" />
</Modularity:ModuleCatalog>
```

```
protected override IModuleCatalog CreateModuleCatalog()
{
    return Microsoft.Practices.Prism.Modularity.ModuleCatalog.CreateFromXaml(
            new Uri("/WpfApplication1;component/ModuleCatalog.xaml", UriKind.Relative)
        );
}
```

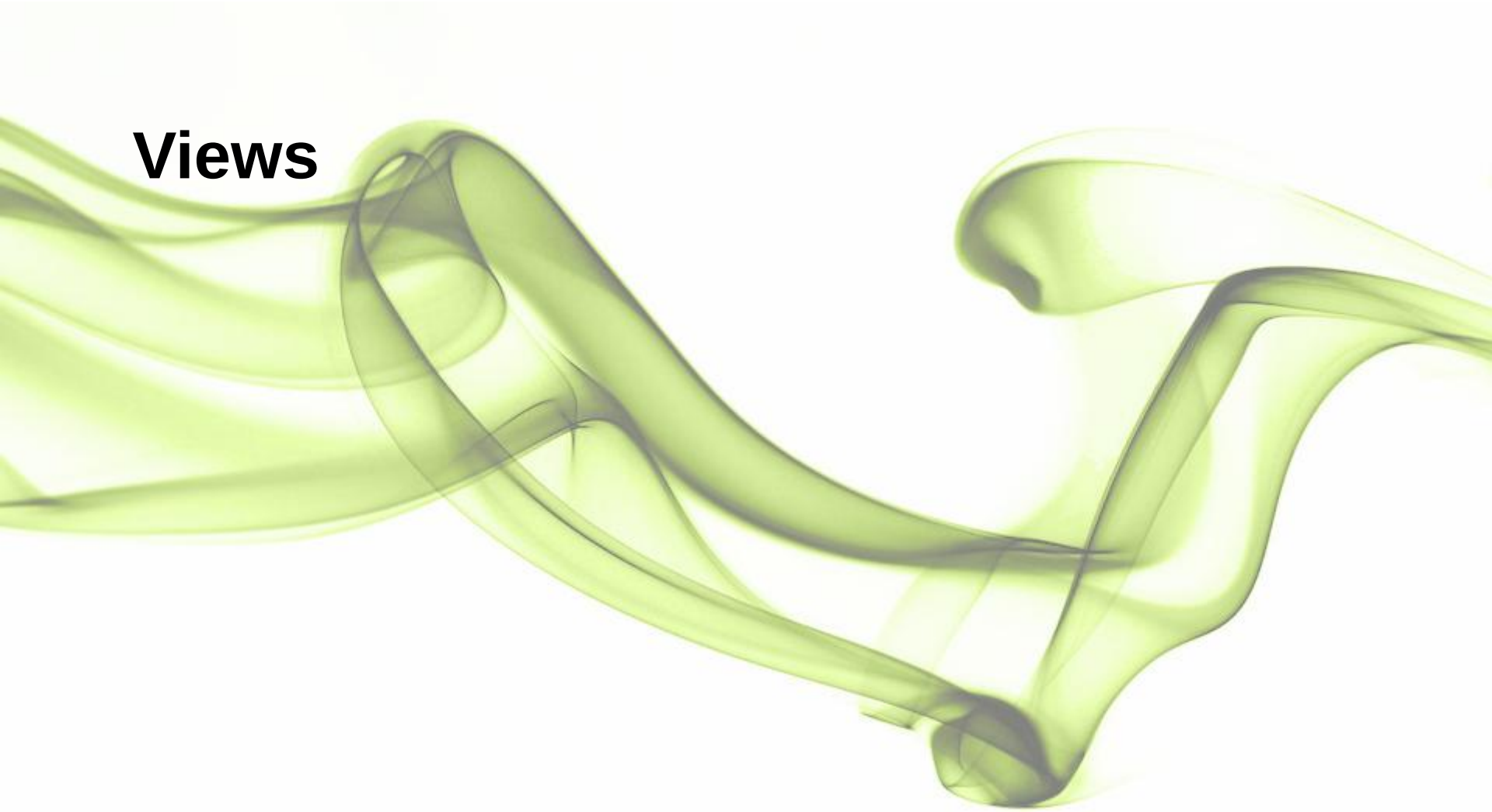
Modules registrieren

- aus app.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="modules"
             type="Microsoft.Practices.Prism.Modularity.ModulesConfigurationSection, Microsoft.Practices.Prism"/>
  </configSections>
  <modules>
    <module assemblyFile="Modules/ModuleA.dll"
            moduleType="ModuleA.ModuleAModule, ModuleA, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"
            moduleName="ModuleA"
            startupLoaded="true" />
  </modules>
</configuration>
```

```
protected override IModuleCatalog CreateModuleCatalog()
{
    return new ConfigurationModuleCatalog();

}
```

Views



Views

- Teil der Benutzeroberfläche
- unabhängig von anderen Views
- kann aus anderen Views bestehen
- kann mehrfach instanziiert werden
- kein Designpattern notwendig, aber Empfehlung für MVVM

Views erstellen

- Interfaces im Infrastructure-Projekt

```
public interface IViewModel
{
}
```

```
public interface IView
{
    IViewModel ViewModel { get; set; }
}
```

- Interface und Implementierung für das ViewModel im Modul erstellen

```
public interface IContentAViewViewModel
    : IViewModel
{
}
```

```
public class ContentAViewViewModel
    : IContentAViewViewModel
{
}
```

```
public partial class ContentView : UserControl, IView
{
    public ContentView(IContentAViewViewModel viewModel)
    {
        InitializeComponent();

        ViewModel = viewModel;
    }

    public IViewModel ViewModel
    {
        get { return (IViewModel) DataContext; }
        set { DataContext = value; }
    }
}
```

Views erstellen

- Typen registrieren

```
public void Initialize()
{
    _container.RegisterType<ToolbarView>();
    _container.RegisterType<ContentView>();
    _container.RegisterType<IContentAViewViewModel, ContentAViewViewModel>();
}
```

Views in Regions registrieren

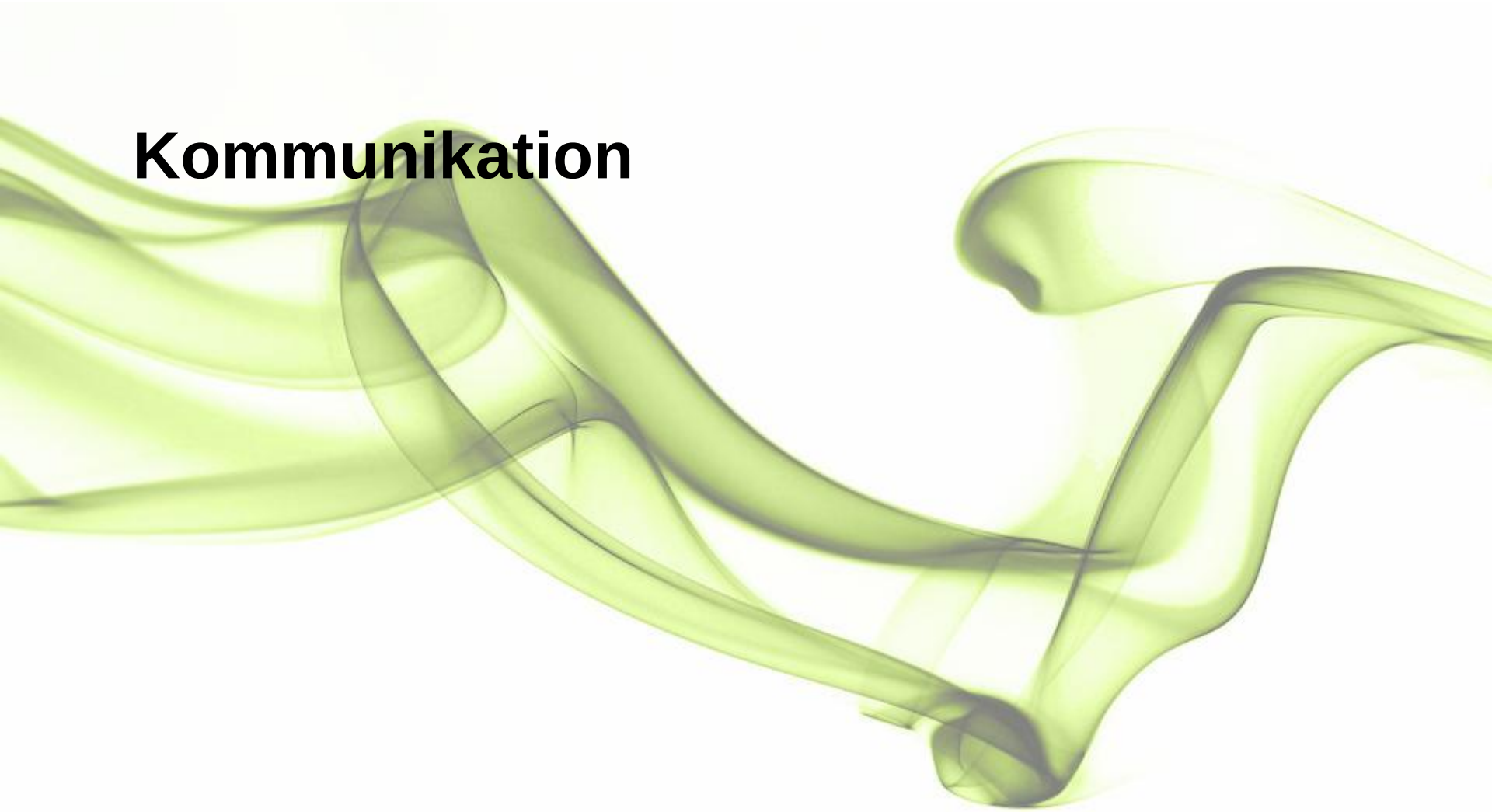
- View Discovery
  - Views werden automatisch zu Regions hinzugefügt
    - RegionManager.RegisterViewWithRegion
- View Injection
  - Views werden programmatisch zur Regions hinzugefügt
    - RegionManager.Region[“RegionName“].Add
    - IRegion.Add
  - ggf. vorher bestehende Views deaktivieren

Views in Regions registrieren

- Beispiel:

```
public void Initialize()
{
    _container.RegisterType<ToolbarView>();
    _container.RegisterType<ContentView>();
    _container.RegisterType<IContentAViewViewModel, ContentAViewViewModel>();

    _regionManager.RegisterViewWithRegion(RegionNames.ToolbarRegion, typeof(ToolbarView));

    var view = _container.Resolve<ContentView>();
    ((IContentAViewViewModel) view.ViewModel).Message = "First View";

    var contentRegion = _regionManager.Regions[RegionNames.ContentRegion];
    contentRegion.Add(view);

    var view2 = _container.Resolve<ContentView>();
    ((IContentAViewViewModel)view2.ViewModel).Message = "Second View";

    contentRegion.Deactivate(view);
    contentRegion.Add(view2);
}
```

Kommunikation



Kommunikation

- Module sind lose gekoppelt
- Kommunikation zwischen Modulen notwendig, um z.B. auf Aktionen zu reagieren
- Möglichkeiten
  - Commanding
  - Event Aggregation
  - Shared Services
  - Region Context

Commanding

- bekanntes Pattern der WPF
- Commands mit Execute und CanExecute
- Arten:
  - RoutedCommand
    - benötigt Event-Handler im Codebehind
  - DelegateCommand
    - verwendet Methoden-Delegates, die aufgerufen werden, wenn das Command ausgelöst wird
    - kein Event-Handler im Codebehind notwendig
    - meist lokaler Scope
    - Arten: DelegateCommand / DelegateCommand<T>

Commanding

- Arten:
  - CompositeCommand
    - meist globaler Scope
    - mit lokalen Commands, die aufgerufen werden, wenn das CompositeCommand ausgelöst wird (z.B. Speichern)

Demo: Command und CompositeCommand

```
public EmployeeViewModel()
{
    SaveCommand = new DelegateCommand(Save, CanSave);
    GlobalCommands.SaveAllCommand.RegisterCommand(SaveCommand);
}

public DelegateCommand SaveCommand;

private void Save()
{
    LastUpdatedDate = DateTime.Now;
}

private bool CanSave()
{
    return true;
}
```

```
<ToolBar>
    <Button Command="{x:Static Infrastructure:GlobalCommands.SaveAllCommand}">Speichern</Button>
</ToolBar>
```

```
public static class GlobalCommands
{
    public static CompositeCommand SaveAllCommand = new CompositeCommand();
}
```

Event Aggregation

- 2 Rollen:
  - Publisher: feuert einen Event
  - Subscriber: abonniert Events
- Service in Prism
  - IEventAggregator
  - unterstützt mehrere Publisher, die den gleichen Event feuern können und mehrere Subscriber, die den gleichen Event abonnieren können
  - unterstützt Filterung von Events
  - Basisklasse für Events: CompositePresentationEvent<T>
    T = Payload, der an den Subscriber gesendet werden soll

Demo: Event Aggregation

```
public class EmployeeUpdatedEvent : CompositePresentationEvent<string>
{
}
```

```
private void Save()
{
    LastUpdatedDate = DateTime.Now;
    _eventAggregator.GetEvent<EmployeeUpdatedEvent>().Publish(string.Format("{0}, {1}", Employee.LastName, Employee.FirstName));
}
```

```
public class StatusbarViewModel : IStatusbarViewModel
{
    private readonly IEventAggregator _eventAggregator;

    public StatusbarViewModel(IEventAggregator eventAggregator)
    {
        _eventAggregator = eventAggregator;
        _eventAggregator.GetEvent<EmployeeUpdatedEvent>().Subscribe(EmployeeUpdated);
    }

    private void EmployeeUpdated(string payload)
    {
        Message = payload + " wurde aktualisiert";
    }

    private string _message = "Ready";
    public string Message...

    public event PropertyChangedEventHandler PropertyChanged;
}
```

Shared Services

- implementiert als Singleton
- in eigenem Modul mit gemeinsamem Interface
- z.B. auflösbar mit einem ServiceLocator

Demo: Shared Services

```
public class ServicesModule : IModule
{
    private readonly IUnityContainer _container;

    public ServicesModule(IUnityContainer container)
    {
        _container = container;
    }

    public void Initialize()
    {
        // register service as singleton
        _container.RegisterType<IEmployeeRepository, EmployeeRepository>(new ContainerControlledLifetimeManager());
    }
}
```

```
private void Save()
{
    int count = _employeeRepository.SaveEmployee(Employee);
    MessageBox.Show(count.ToString());
}
```

```
namespace PrismDemo.Infrastructure
{
    public interface IEmployeeRepository
    {
        int SaveEmployee(Employee employee);
    }
}
```

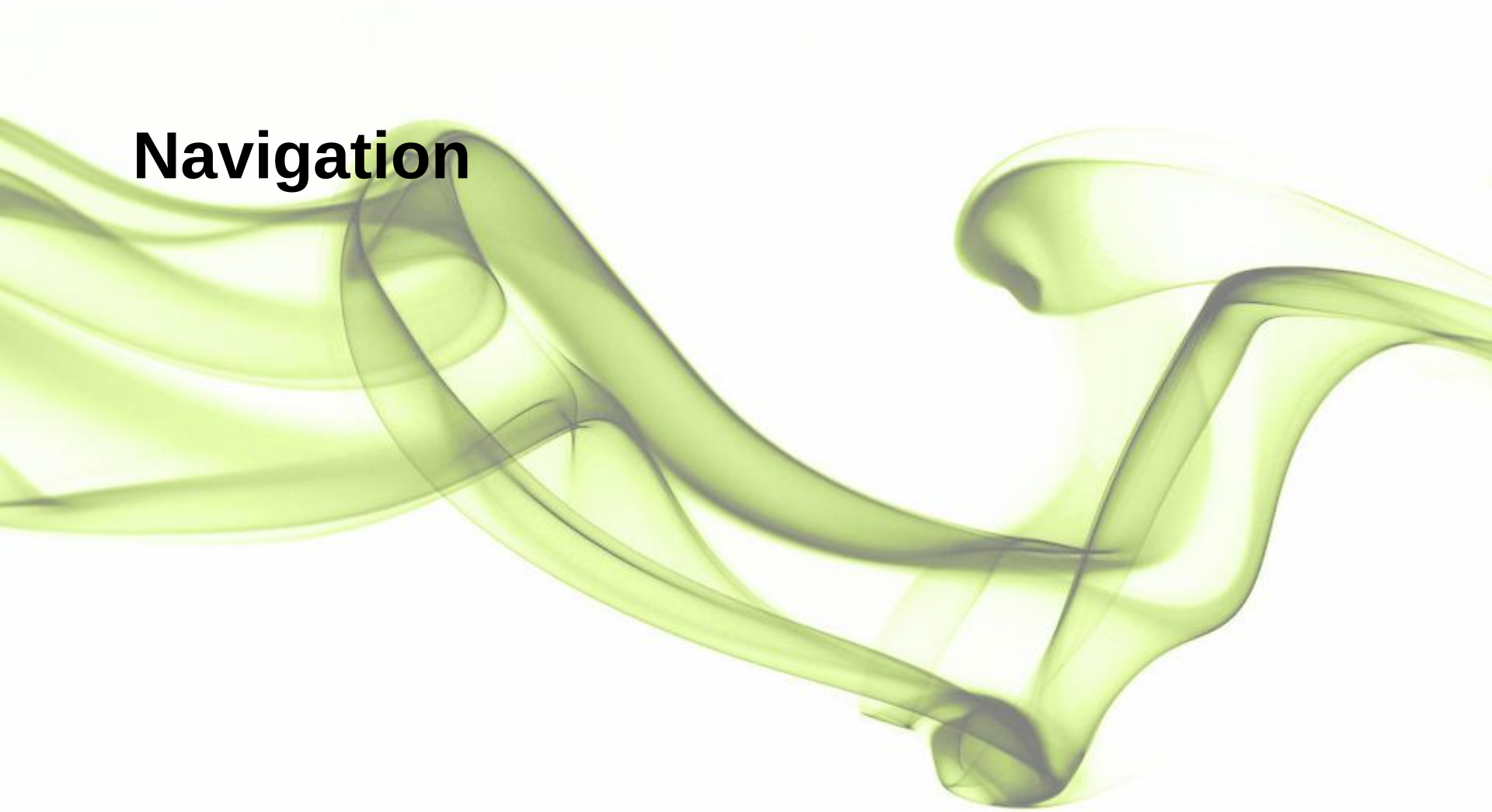
Region Context

- Möglichkeit, Objekte zwischen dem Region Host und den Views in der Region zu teilen (z.B. in Master-Detail-Szenarien)
- View muss ein DependencyObject sein

```
<ContentControl prism:RegionManager.RegionName="DetailsRegion"
                prism:RegionManager.RegionContext="{Binding SelectedItem, ElementName=employeeList}"/>
```

```
public EmployeeView(IEmployeeViewModel viewModel)
{
    InitializeComponent();

    ViewModel = viewModel;

    RegionContext.GetObservableContext(this)
        .PropertyChanged += (s, e) =>
                            {
                                var context = (ObservableObject<object>) s;
                                var selectedEmployee = (Business.Employee) context.Value;
                                ((IEmployeeViewModel)ViewModel).Employee = selectedEmployee;
                            };
}
```

Navigation



Statusbasierte Navigation

- Views werden nicht ersetzt, sondern aktualisiert
  = Statuswechsel auf existierenden Controls
- findet Verwendung, wenn
  - gleiche Daten anders angezeigt werden sollen
    - z.B. Ansicht als Liste oder Icons
    - Austausch des ItemTemplates mit Triggers und Styles
  - View ihr Aussehen basierend auf einem Property im ViewModel ändern soll
    - z.B. IsBusy mit BusyIndicator
  - Aufgaben auf den Daten der View durchgeführt werden sollen
    - z.B. Edit-Action mit ChildWindow
- UX-Vorteil: View wird nicht verlassen

Viewbasierte Navigation

- View in einer Region wird durch eine andere View ersetzt
- implementiert durch Aufruf von RequestNavigate auf der Region oder dem RegionManager

- Views müssen als object im Container registriert werden
- Navigation-Callbacks möglich

- Interface INavigationAware ermöglicht Überwachung von NavigatedFrom und NavigatedTo (für View und ViewModel)

Viewbasierte Navigation

- Parameter
  - mit Hilfe von UriQuery

```
if (person != null)
{
    var uriQuery = new UriQuery();
    uriQuery.Add("To", person.Email);

    var uri = new Uri(typeof(EmailView).FullName + uriQuery, UriKind.Relati
    _regionManager.RequestNavigate(RegionNames.ContentRegion, uri);
}
```

  - abrufbar im NavigationContext von INavigationAware.OnNavigatedTo

Viewbasierte Navigation

- Lifecycle bestehender Views
  - IRegionMemberLifeTime ermöglicht, dass Views beim Deaktivieren entfernt werden
    - KeepAlive = false => Freigabe für GC
  - Views werden nur dann weiterverwendet, wenn IsNavigationTarget = true

```
public bool IsNavigationTarget(NavigationContext navigationContext)
{
     string id = navigationContext.Parameters["Id"];
     return this.currentCustomer.Id.Equals(id);
}
```

Viewbasierte Navigation

- Abbrechen der Navigation
  - IConfirmNavigationRequest (implementiert INavigationAware)

```
public void ConfirmNavigationRequest(NavigationContext navigationContext,
                                     Action<bool> continuationCallback)
{
     //by default we allow navigation
     bool result = true;

     //prompt user for desired action
     MessageBoxResult messageResult = MessageBox.Show("Save", "Save Object?",
                                                      MessageBoxButton.YesNoCancel);

     if (messageResult == MessageBoxResult.Yes)
          CurrentCustomer.Save(); //perform save
     else if (messageResult == MessageBoxResult.Cancel)
          result = false; //don't allow navigation

     continuationCallback(result);
}
```

Viewbasierte Navigation

- Navigation-Journal (IRegionNavigationJournal)
  - erlaubt Vorwärts- und Rückwärts-Navigation (GoBack, GoForward)
  - nur für Views, die vom Region Navigation Service hinzugefügt wurden (nicht für per View Discovery oder View Injection hinzugefügte Views)
  - stackbasiert
  - NavigationContext.NavigationService.Journal

Vielen Dank für Ihre Aufmerksamkeit.

Unique Software e.K.
Hans-Sachs-Str. 38
01129 Dresden

Mail: info@uniquesoftware.de
Web: http://www.uniquesoftware.de