FHWS

Hochschule für angewandte Wissenschaften
Fachhochschule Würzburg-Schweinfurt

Seminararbeit

Das MVC - Entwurfsmuster und Erweiterungen

vorgelegt an der
Hochschule für angewandte Wissenschaften Würzburg-Schweinfurt
in der Fakultät Informatik und Wirtschaftsinformatik
FWPM Design Patterns

bei Professor:	Prof. Eberhard Grötsch
Abgabetermin:	23. Mai 2013
Eingereicht von:	Florian Beckh Christian Petry Johannes Krenig 5109038

Inhaltsverzeichnis

1 MVVM

Während die Ursprünge des Model View Controller (MVC)-Entwurfsmusters in den 1980er Jahren liegen und das Model View Presentation (MVP)-Entwurfsmuster erst in den 1990 Jahren zum ersten Mal beschrieben wurde, ist das Model View View Model (MVVM) noch relativ jungen Alters. Das MVVM ist zudem anders als die beiden anderen Entwurfsmuster das spezifischste Muster, denn während MVC und MVP relativ geringe Voraussetzungen für ihre Verwendung haben, ist MVVM speziell für morderne Technologien konzipiert. So setzt die Verwendung des MVVM den Einsatz von Microsofts Windows Presentation Foundation (WPF) oder Silverlight zwar nicht unbedingt voraus, ist aber ratsam. Trotz dieser Einschränkung, ist es kein Fehler sich mit diesen Muster zu beschäftigen, schließlich werden mittlerweile ein Großteil aller UIs, die auf .NET-Framework basieren damit erstellt. In diesem Kapitel wird daher zuerst auf die Verbindung WPF und MVVM eingegangen, bevor eine allgemeine Problemstellung aufgezeigt wird, die eine auf dem MVVM-Entwurfsmuster basierende Anwendung löst. Die Konsequenzen die aus dem Einsatz des MVVM sich ergeben und ein praktisches Beispiel schließen dieses Kapitel ab.

1.1 WPF und MVVM-Entwurfsmuster

1.1.1 WPF

Mit .NET Framework 3.0 wurde 2006 WPF ein mächtes Grafikframework für Windows-Anwendungen eingeführt. Das rendern und zeichnen von Inhalten findet mittels DirectX auf der Graphic Processor Unit (GPU) statt, dadurch wird es dem GUI-Designern ermöglicht optisch ansprechende Benutzeroberflächen zu designen, bei trotzdem geringer Belastung der Central Processor Unit (CPU). Wohltuend ist die klare Trennung der Präsentations- und Geschäftslogik: Während die Geschäftslogik in Programmcode geschrieben ist, wird die Präsentation mittels der Auszeichnungssprache Extensible Application Markup Language (XAML) deklariert. XAML ist wie der Name schon impliziert an XML angelehnt (siehe Listing 1.1). Die Idee eine Auszeichnungssprache als Metasprache für die Benutzeroberfläche zu verwenden hatte sich zuvor bereits bei HTML bewährt.

```
<Page
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="ExampleNamespace.ExamplePage">
  <Button Click="Button_Click" >Click Me!</Button>
</Page>
```

Listing 1.1: Beispielcode XAML

Aktionen, die z. B. bei Klicken auf einen Button aufgerufen werden können, werden Codebehind geschrieben. Hierbei handelt es sich um normalen ausführbaren Programmcode in einer beliebigen .NET kompilierbaren Programmiersprache (siehe Listing 1.2).

```
namespace ExampleNamespace
{
  public partial class ExamplePage
  {
    void Button_Click(object sender, RoutedEventArgs e)
    {
      Button b = e.Source as Button;
      b.Foreground = Brushes.Red;
    }
  }
}
```

Listing 1.2: Beispielcode Codebehind in C#

Ein wichtiger Bestandteil der WPF, für die Nutzung des MVVM, ist die Definition und Bereitstellung des Interfaces “ICommand“.

Die wichtigsten Eigenschaften einer WPF-Anwendung im Überblick:

- Design mit der Auszeichnungssprache XAML, alternativ auch ausprogrammiert möglich.
- Unterstützung von 2D und 3D Grafiken
- Die Ausgabe ist vektorbasiert anstatt pixelbasiert. Daraus erfolgt eine bessere Skalierbarkeit der Bildschirmausgabe.
- Vielfältige Datenbindungsmöglichkeiten
- Rendern und Grafikberechnung auf der GPU anstatt CPU

Die strikte Trennung von Oberflächengestaltung und Code ermöglicht es, dass Programmentwickler und Oberflächendesigner vollkommen unabhängig voneinander Code und Benutzeroberfläche entwickeln können.

1.1.2 MVVM

Das MVVM Entwurfsmuster wurde von John Gossman als Variante des MVP Entwurfsmuster auf seinem Blog vorgestellt. Hintergrund war die fortschreitende Entwicklung moderner Benutzerinterface-Frameworks, wie WPF und Silverlight von Microsoft. Die Abkürzung MVVM steht wie bei MVC und MVP für die Komponenten des Entwurfsmusters (Siehe Abbildung 1.1). Die Langform Model View ViewModel ist auf den ersten Blick ungewöhnlich und bedarf der Erklärung.

- Model: Datenzugriffsschicht für die Inhalte, dem Benutzer angezeigt und von ihm manipuliert werden. Dazu benachrichtigt es über Datenänderungen und führt eine Validierung der vom Benutzer übergebenen Daten durch. Hierdurch wird vor allem in der View der Code-Behind minimiert.

- View: Alle durch die GUI angezeigten Elemente. Es bindet sich an Eigenschaften des ViewModel, um Inhalte darzustellen und zu manipulieren sowie Benutzereingaben weiterzuleiten. Durch die Datenbindung ist die View einfach austauschbar und ihr Code-Behind gering.

- ViewModel: beinhaltet die UI-Logik (Model der View) und dient als Bindeglied zwischen View und obigem Model. Einerseits tauscht es Information mit dem Model aus, ruft also Methoden oder Dienste auf. Andererseits stellt es der View öffentliche Eigenschaften und Befehle zur Verfügung. Diese werden von der View an Steuerelemente gebunden, um Inhalte auszugeben bzw. UI-Ereignisse weiterzuleiten. Insgesamt wird CRUD ermöglicht. Das ViewModel darf dabei keinerlei Kenntnis der View besitzen.

Die vom MVVM genutzte funktionale Trennung und Datenbindung des MVC wird dazu genutzt, die bereits beschriebene lose Kopplung zu erreichen. Diese ist jedoch im MVVM um einiges stärker verwirklicht als in MVC und MVP.

Aufgrund der Tatsache, dass Sie beim MVVM die Logik und das Command in der ViewModel-Klasse haben möchten, haben sich bei diesem Entwurfsmuster Commands wie das ActionCommand gegenüber den RoutedCommands durchgesetzt. Ein ActionCommand erlaubt beim Instanziieren direkt die Angabe eines Delegates, der auf eine Methode im ViewModel zeigen kann. MVVM in Verbindung mit WPF verwendet üblicherweise eine Implementierung mit Delegates basierend auf ICommand-Interfaces.

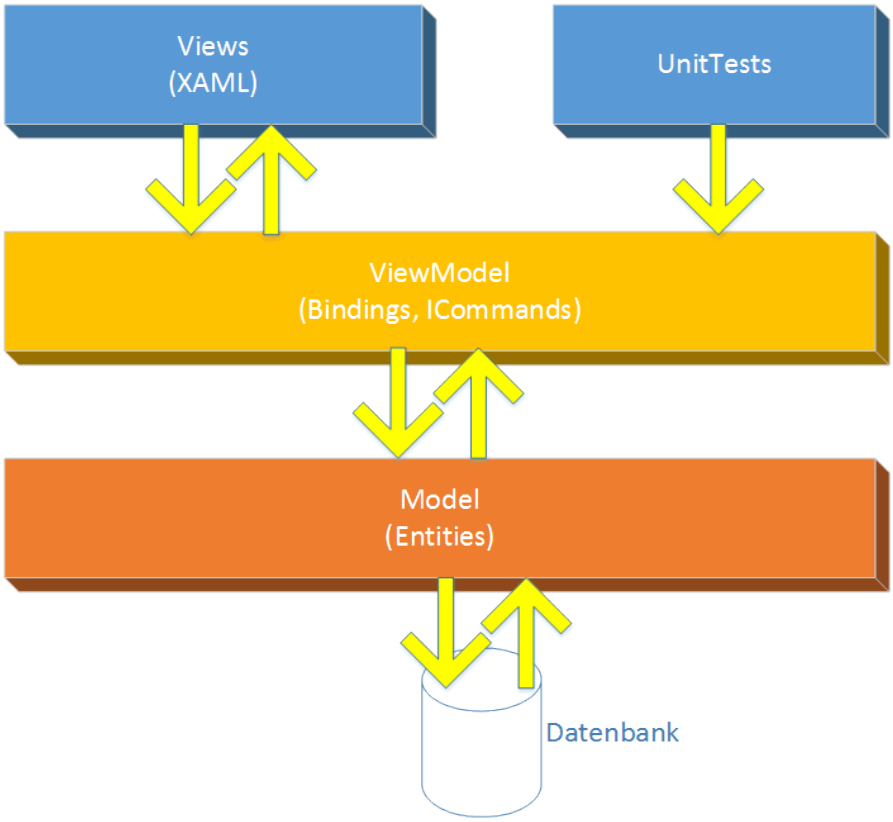

Abbildung 1.1: MVVM-Entwurfsmuster

1.2 Problem

Inetec produces multi-axis manipulators for various nuclear power plant non-destructive testing (NDT) inspection services. Manipulators are tele-operated at a distance over a network where the operator station needs to have feedback about the manipulator position, attitude, status and health. Operator can move the axis in manual mode, perform and specify a coordinated multi-axis drive mode which is used for actual inspection when data acquisition is being performed. Manipulator position is displayed as position of various manipulator axes that are calibrated; however position is also determined by the operator looking at the various camera live video feeds. Software overlays position and other data to the video feed as well, providing better operator experience. Inspection test data is acquired from various ultrasound (UT) and Eddy current (ET) probes, driven by the manipulator over the surface of interests, thus one module is synchronization module between Inetec manipulator control software and UT and ET instruments.

Each manipulator produced on a custom basis for a particular customer shares a lot of commonality in terms of hardware, electronics and software but is also customized for a particular customer according to their specifications and expected needs. Often times, there are requests to modify the GUI to suit specific customer need, add additional control

component for a specific custom application or make several GUIs to be used with the same equipment depending on the actual operator role. Thus, it is essential that software is flexible enough to accommodate those changes while not affecting other software code parts to prevent introducing the bugs as a side-effect of the changes. Lastly, in today's market of this specific industry, all parts of the system including software need to be fully localized to the native language and it is essential that technology used to develop such software support this requirement seamlessly.

1.3 Lösung

WPF was designed to greatly decouple GUI from the rest of the code to a far greater extent than it was possible using MVC/MVP pattern for GUI development. At Inetec, we design all GUI components in XAML and declarative nature of XAML speeds up GUI design process. Using WPF and XAML to define a GUI we can experiment easily and change/redefine the GUI quicker. Our designers can develop GUI design and work with the customer and application departments to come up with a intuitive design while development team focuses on logic and other modules of the software application. Lastly, WPF support for localization in terms of auto sizing of GUI widgets and layouts helps us in localizing our software, which was not possible using .NET WinForms where a widget had to be resized manually if localized text did not fit. Using MVVM pattern greatly improved our productivity and drastically reduced side-effect errors when a modification had to be done. With MVVM pattern we were able to produce self contained modules consisting of Model and ViewModel parts for a certain object in our system (axis, motor, solenoid), fully test it using testing harnesses. ViewModel exposes properties and command objects a GUI (a View in the MVVM pattern) can display or act upon. GUI changes or modifications do not require any changes of the ViewModel that has logic, etc and View does not have any code behind – it is pure XAML having only GUI look design declaration and data binding specification. Thus, to customize a GUI for a specific customer we can only change the XAML portion of the application to expose or hide a certain property or command binding in case more or less data is needed. To change physical layout and design XAML is also easily changed. To do a more involved change, we can create another ViewModel and corresponding XAML and other parts of the system do not have to be modified because XAML can use DataTemplate to select dynamically a View for a certain ViewModel object instance. Thus, main window XAML is minimally changed to register the new ViewModel and rest of the application code remains unchanged. It should be noted that a bigger change may require changing of the other ViewModels and performing their testing again, but at Inetec using WPF and MVVM pattern we have reduced our time and effort to do changes to the GUI by about 80application was developed and used. Effort in doing the localization and making sure all GUI elements display correctly was reduced drastically to a similar percentage.

1.4 Konsequenz

Der Nutzen des MVVM besteht in der Entkopplung des Designs der Userinterfaces von der Programmierung der Logik. Diese ist jedoch erkauft durch einen Overhead an Konzeptions- und Codierarbeit. Das MVVM lohnt sich daher nicht für jedes Projekt. Als Faustregel gilt, dass MVVM die falsche Wahl ist, wenn gilt:

- kleines Projekt
- eine oder wenige Views

Hingegen kann MVVM als gute Wahl gelten, wenn gilt:

- großes Projekt
- viele Views
- neue Views können hinzu kommen
- Einsatz von WPF oder Silverlight

1.5 Beispiel

This section briefly describes a 3-axis manual, for example an XYZ table, manipulator using MVVM pattern. Finished application would look like in Fig 3. A main window can be seen along with a motion axis section having 3 axes each with its own controls and data. Below we have a general system health monitor and on the right we have scan plan management section which is currently hidden due to the application state. Following MVVM pattern, Main window has its corresponding ViewModel that has a property list of AxesViewModels object instances and one ViewModel for system instance. For each ViewModel, there is a custom WPF user control created which corresponds to a View and is bound to properties of its particular ViewModel. For axis, textbox next to position label is bound to a position property of the ViewModel, etc. In XAML on the application level, in a data template each ViewModel is registered to use a particular custom WPF user control. Thus, in MainWindow when <ItemsControl> GUI Widget is bound to property which is a list of AxisViewModels, it will populate its items list with instances of Views defined in the data template and would set the data context of View to a particular item in the list it was bound to – showing 3 axes (items) as in Fig 3. Figure 3. Sample application screenshot showing GUI structure. As ViewModel data changes, for example axis position changes, View will display correct position in the textbox automatically due to binding. Thus, once ViewModel is created along with corresponding user control implementing the View, things work automatically in WPF and MVVM framework. Lastly, in the above example it can be seen that leftmost axis X has different frame around it which shows that axis is enabled. Such more expressive GUI was done purely in XAML by using data triggers on certain bound properties. Similarly button text changes from Enable axis to Disable axis, AxisError is painted red and not light gray, etc. Traditionally in WinForms development there should be code written to accomplish those changes whereas in WPF this can be done by a GUI designer in XAML, without touching underlying code. Thus, WPF and MVVM can provide for a rapid prototype development to be used quickly with a simple and then GUI can be further refined in function, visual cues and graphic design with frozen and tested ViewModel code. This has proven to be greatly beneficial at Inetec in custom robot software design, along

Literaturverzeichnis

Listings

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

CPU Central Processor Unit 2

GPU Graphic Processor Unit 2

MVVM Model View View Model 1

MVC Model View Controller 1

WPF Windows Presentation Foundation 1

MVP Model View Presentation 1

XAML Extensible Application Markup Language 2