Carsten Kuhn, Danny Kautzsch, Matthias Jauernig
Lehrveranstaltung „Compilerbau (Aufbaukurs)“
Prof. Waldmann, Fb IMN, HTWK Leipzig

Leipzig, 01.02.2008

Projekt „Xaml-Konverter“

Aufgabenbeschreibung

Mit *Xaml* (Extensible Application Markup Language) wurde in .NET3.0 ein Konzept eingeführt[1], mit dem GUIs durch eine *Xml*-ähnliche Sprache beschrieben werden können. *Xaml* ermöglicht so die Trennung von Oberfläche und Programmlogik und stellt die Basis für komfortable und reichhaltige GUI-Editoren dar.

Mit dem *XamlKonverter* wurde im bearbeiteten Projekt eine Möglichkeit geschaffen, *Xaml* in C#-Code umzuwandeln, um die GUI beispielsweise in *Xaml* zu erstellen, aber in C# zu halten oder Äquivalente zu *Xaml*-Codestücken in C# betrachten zu können. Der *XamlKonverter* ist u.a. für die dynamische Gestaltung von Oberflächen interessant, da Microsoft bisher nur wenige Codebeispiele in C# dokumentiert hat.
Im Anhang A1 ist eine Abbildung der Programmoberfläche und in A3 ein Beispiel für in C# umgewandelten Xaml-Code enthalten.
Ziel des Projekts war es dabei nicht, alle *Xaml*-Objekte und Attribute zu behandeln. Vielmehr sollten die von *Xaml* gelieferten Möglichkeiten der hierarchischen Beschreibung von GUI-Elementen ermittelt und umgesetzt werden. Zu Testzwecken wurde die Behandlung verschiedener Klassen (s. Anhang A2) der *Windows Presentation Foundation (WPF)*[2] mit ausgewählten Eigenschaften implementiert. Dabei wurde deutlich, dass der erstellte Konverter nicht nur viele sprachliche Mittel von *Xaml* abdeckt, sondern auch über eine sehr einfache Erweiterbarkeit um weitere Elemente und Schnittstellen verfügt.

Allgemeine Vorgehensweise

Die Spezifikation von *Xaml* ermöglicht die Verknüpfung mit Programmcode. Dafür wird das definierte Element `<x:Class>` verwendet. Diese Einbettung ist jedoch nur für Projekte vorgesehen und erfordert daher eine Filterung beim Laden von *Xaml*-Code aus Dateien. So wird zunächst ein selbst geschriebener Filter auf den einzulesenden *Xaml*-Code angewendet. Neben `<x:Class>` werden in diesem Schritt noch weitere Elemente gefiltert, die nur für .NET-Projekte vorgesehen sind.
Zum Einlesen einer gefilterten *Xaml*-Datei wird die *XamlReader*-Klasse aus dem Namensraum *System.Windows.Markup* zur Verfügung gestellt. Diese Klasse ist für die Konvertierung von *Xaml* in UI-Objekte zuständig. Dabei erfolgt durch die XamlReader-Klasse eine Syntaxprüfung des einzulesenden Xaml-Codes. Eine Vorschau der UI lässt sich ebenfalls mit dem erzeugten Objekt realisieren.
Anschließend wird das *UI*-Objekt wieder in *Xml* serialisiert. Dadurch erhält man den *Xaml*-Code als Baumstruktur, welche traversiert und verarbeitet werden kann.

Da der *Xaml*-Code noch ungeordnet ist und Attributwerte teilweise konvertiert werden müssen, wird im nächsten Schritt eine wohl definierte Baumstruktur erzeugt. In dieser sind alle Informationen des *Xaml*-Codes enthalten und die Werte von *Xaml*-Attributen werden in diesem Baum C#-konform gehalten. Für die Konvertierung von Attributwerten werden einige Klassen in .NET3.0 angeboten. Diese lassen sich für eigene

[1] In .NET3.5 um weitere Regeln erweitert.
[2] WPF ist Bestandteil von .NET3.0. Dabei handelt es sich um eine API zur Erstellung von anspruchsvollen 2D- und 3D-Benutzeroberflächen.

Konverter-Klassen nutzen, die erforderlich sind, da *Xaml* z.B. verschiedene Einheiten (px, cm, pt) unterstützt und diese in *double*-Werte konvertiert werden müssen.

Im letzten Schritt wird der eigentliche C#-Code durch einfache Traversierung des Baumes erzeugt. Als Ergebnis wird eine Klasse „GUI“ ausgegeben, welche den C#-Code inkl. notwendiger using-Anweisungen und erforderlicher Klassen-Variablen enthält. Bei der Traversierung wird dabei zunächst eine lineare Liste gekapselter Anweisungen erzeugt (es findet eine Linearisierung des Baumes statt), die dann durch einfaches Abarbeiten in einen Code-String umgewandelt wird.

Baumbeschreibung

Der erzeugte interne Baum bildet im Wesentlichen die im *Xaml*-Code enthaltene Hierarchie ab. Die grundlegende Idee ist es, dass für jedes Element des *Xaml*-Codes ein eigenständiger Baumknoten gehalten wird. Grundsätzlich werden dabei zwei *Xaml*-Objekte unterschieden: Elemente (`<Tag />`) und Attribute (`<Tag Attributname="Wert" />`). Entsprechend gibt es für diese 2 Objekte unterschiedliche Arten von Baumknoten. *Element-Knoten* können Attribut-Knoten sowie weitere Element-Knoten als Kinder enthalten. Ebenso haben sie einen Namen (gibt die C#-Klasse an, z.B. „`Button`“) und einen Variablennamen, durch den sie im resultierenden C#-Code reflektiert werden (z.B. „`button1`“). Dieser Variablenname ist eindeutig für den erzeugten C#-Code. Ein *Attribut-Knoten* gehört zu genau einem Element-Knoten und hat den Namen des Attributs sowie den konvertierten Attribut-Wert (als C#-konformer Ausdruck) als Inhalt.

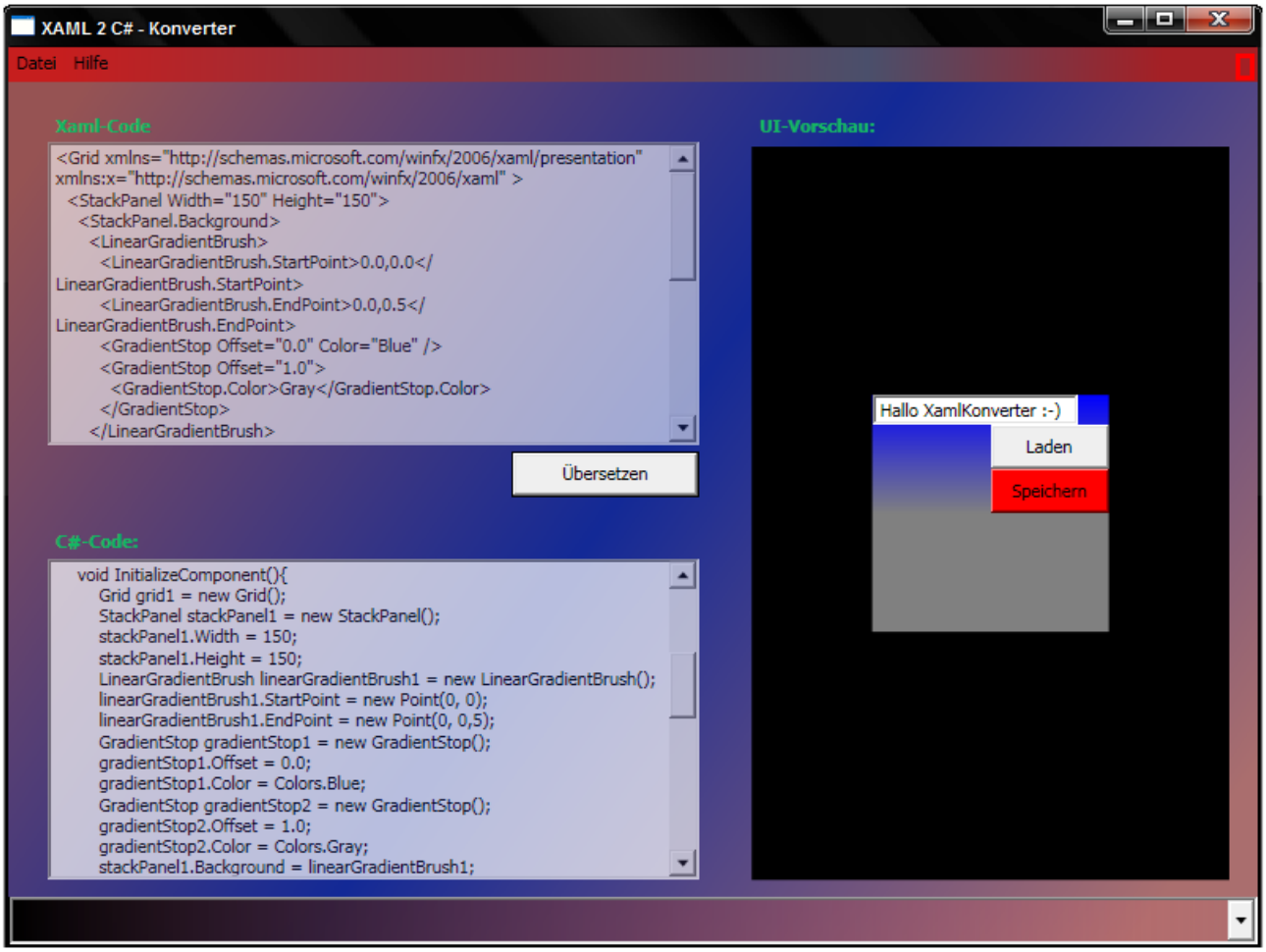
Ein dritter Baumknotentyp ist eine Art „Zwitter“ aus Element-Knoten und Attribut-Knoten und tritt auf, wenn ein Attribut im *Xaml*-Code als eigenes Tag steht und weitere Element-Knoten als Kinder besitzt. Dies kann z.B. dann der Fall sein, wenn es sich bei dem Attribut im C#-Code um eine Klasseneigenschaft handelt, die eine *Collection* darstellt und der somit weitere Objekte zugewiesen werden können.

Hierarchiebildung

Ein Basis-Konzept des *XamlKonverter*‘s bilden die Node-Generatoren und die darauf basierende Abbildung der Hierarchie von GUI-Elementen in C#. `NodeGenerator` ist eine abstrakte Klasse, welche durch konkrete Implementierungen vererbt werden muss. Die Idee hinter diesem Konzept ist, für jedes durch den *XamlKonverter* zu behandelnde GUI-Element (`Button`, `Grid`, etc.) eine eigene konkrete `NodeGenerator`-Vererbung zu erzeugen. Für die Basisklassen der GUI-Elemente (z.B. `ButtonBase`, `ContentControl`, `Control` etc. bei `Button`) werden weiterhin abstrakte `NodeGenerator`-Klassen erstellt, wobei diese entsprechend ihrer Hierarchie in C# auch im *XamlKonverter* angeordnet werden. Vergleicht man die C#-Hierarchie der GUI-Elemente mit dem hier abgebildeten Klassendiagramm, so erkennt man direkt die 1:1-Abbildung (siehe auch Anhang A4). Jeder Knoten im erstellten Baum beinhaltet einen *NodeGenerator*, durch den er erzeugt wurde und der den „Typ“ des in dem Knoten gespeicherten GUI-Elements angibt. Mit Node-Generatoren können zum einen eindeutige Variablennamen über einen fortlaufenden Index erzeugt werden, zum anderen lässt sich damit aus der Liste der Attribute eines *Xaml*-Knotens eine Liste von konvertierten Attributknoten im Baum erstellen.
Dieser Aufwand der Hierarchiebildung wird aus Gründen der größtmöglichen Vermeidung von Redundanzen betrieben. In C# sind viele Properties nicht direkt in den GUI-Elementen, sondern in ihren Basisklassen definiert. So ist es auch im *XamlKonverter*: die Behandlung der Attribute, für die eigene Baumknoten erstellt werden, findet dort statt, wo diese auch in C# definiert sind – meist in den abstrakten Basisklassen (Node-Generatoren). Die Hierarchie verhilft also dazu, dass Attribute in Basisklassen von Objekten nicht mehrfach behandelt werden müssen.

Anhang

A1) Programmoberfläche:



A2) Implementierte GUI-Elemente

```
Button
CheckBox
ComboBox
ComboBoxItem
GradientStop
Grid
GroupBox
Label
LinearGradientBrush
ListBox
ListBoxItem
Menu
MenuItem
RadioButton
SolidColorBrush
StackPanel
TabControl
TabItem
TextBox
ToolBar
```

A3) Beispiel für Xaml-Code und erstellten C#-Code:

Xaml:

```
<Grid xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
  <StackPanel Width="150" Height="150">
    <TextBox IsReadOnly="true" Width="130">Hallo XamlKonverter :-)</TextBox>
    <Button Name="Button1" Width="75" Height="28">Laden</Button>
    <Button HorizontalAlignment="Right">
      <Button.Width>75</Button.Width>
      <Button.Height>28</Button.Height>
      <Button.Background>
        <SolidColorBrush>Red</SolidColorBrush>
      </Button.Background>
      Speichern
    </Button>
  </StackPanel>
</Grid>
```

C#:

```
using System;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;

class GUI {

    Button Button1 = new Button();

    public GUI() {
        InitializeComponent();
    }

    void InitializeComponent(){
        Grid grid1 = new Grid();
        StackPanel stackPanel1 = new StackPanel();
        stackPanel1.Width = 150;
        stackPanel1.Height = 150;
        TextBox textBox1 = new TextBox();
        textBox1.Text = "Hallo XamlKonverter :-)";
        textBox1.IsReadOnly = true;
        textBox1.Width = 130;
        Button1.Content = "Laden";
        Button1.Name = "Button1";
        Button1.Width = 75;
        Button1.Height = 28;
        Button button1 = new Button();
        button1.Content = "Speichern";
        button1.HorizontalAlignment = HorizontalAlignment.Right;
        button1.Width = 75;
        button1.Height = 28;
        SolidColorBrush solidColorBrush1 = new SolidColorBrush();
        solidColorBrush1.Color = Colors.Red;
        button1.Background = solidColorBrush1;

        stackPanel1.Children.Add(textBox1);
        stackPanel1.Children.Add(Button1);
        stackPanel1.Children.Add(button1);
        grid1.Children.Add(stackPanel1);
    }

}
```

A4) Teil der NodeGenerator-Hierarchie: