Bachelorarbeit

Portierung und Revision einer Steuerelementebibliothek basierend auf dem Microsoft .Net Framework zum Microsoft .Net Compact Framework für ein GUI von Windows CE Geräten.

Baum, Daniel

Geboren am 15. Dezember 1984 in Lichtenstein

Studiengang Informatik

Schwerpunkt: Systementwicklung

Westsächsische Hochschule Zwickau
Fakultät Physikalische Technik / Informatik
Fachgruppe Informatik

Betreuer, Einrichtung:	Prof. Dr. Georg Beier, WH Zwickau Dipl.-Ing. Matthias Päßler, Mühlbauer AG Stollberg Dipl.-Ing. (FH) Ingo Kunz, Mühlbauer AG Stollberg
Abgabetermin:	03. November 2010

Selbständigkeitserklärung gem. § 22 Absatz 5 BPO

Hiermit versichere ich, Daniel Baum, dass ich die vorliegende Bachelorarbeit mit dem Titel,

„Portierung und Revision einer Steuerelementebibliothek basierend auf dem Microsoft .Net Framework zum Microsoft .Net Compact Framework für ein GUI von Windows CE Geräten“

selbständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

....
Ort, Datum Unterschrift

Sperrvermerk

Die vorliegende Bachelorarbeit beinhaltet interne vertrauliche Informationen der Firma Mühlbauer AG. Die Weitergabe des Inhaltes der Arbeit im Gesamten oder in Teilen ist grundsätzlich untersagt. Es dürfen keinerlei Kopien oder Abschriften – auch in digitaler Form – gefertigt werden. Ausnahmen bedürfen der schriftlichen Genehmigung der Firma Mühlbauer AG.

....
Ort, Datum Unterschrift

Autorenreferat

Im Rahmen dieser Arbeit werden der Weg und die entstanden Steuerelemente bei der Portierung einer Steuerelementebibliothek vom .Net Framework zum .Net Compact Framework beschrieben.

Dabei ist die Portierung dafür gedacht, komplette Rechner, welche teilweise nur zu Darstellung eines Graphical User Interfaces (Grafische Benutzeroberfläche) dienen, durch eine preiswertere Steuerplatine zu ersetzen.

Es werden Informationen zu den Unterschieden zwischen dem .Net Framework und dem .Net Compact Framework dargestellt. Anschließend wird die aktuelle Steuerelementebibliothek auf Funktionsweise und Aufbau untersucht.

Den Hauptteil der Arbeit nimmt die Entwicklung einer neuen Steuerelementebibliothek auf Basis des .Net Compact Framework für Windows CE Systeme ein. Hier werden die einzelnen Steuerelemente, der aktuellen Bibliothek untersucht und auf das .Net Compact Framework portiert und angepasst. Es werden auch Probleme bei der Darstellung von Gradienten und Transparenz mit dem .Net Compact Framework und Windows CE erläutert.

Ziel der Arbeit ist es, das Aussehen und die Funktionalitäten der vorhandenen Bibliothek zu portieren und gegebenenfalls neue Steuerelemente zu entwickeln. Die Notwendigkeit besteht darin, dass das künftige Grafische User Interface für Windows CE genauso aussieht und arbeitet wie das aktuelle für Windows XP.

Vorwort

Diese Bachelorarbeit wurde zwischen dem 16.08.2010 und dem 03.11.2010 in Zusammenarbeit mit der Firma Mühlbauer AG erstellt.
Die Mühlbauer AG ist ein weltweit agierender Berater und Hersteller von schlüsselfertigen Produktlösungen im Bereich Smart Cards, Smart Labels, Halbleiter-Back-End und Vision-Systeme. Der Tätigkeitsbereich des Unternehmens erstreckt sich weiterhin auf die Datenerfassung und Datenverifikation, sowie die Herstellung von Präzisionssystemen und OEM-Komponenten.

Ich möchte mich an dieser Stelle bei allen bedanken, die mich bei dieser Arbeit unterstützt haben.

Besonderer Dank gebührt…

… Herrn Dipl.-Ing. Tino Henze und Herrn Dipl.-Ing. Ingo Kunz, für die fachliche Betreuung seitens der Mühlbauer AG und die Ermöglichung der Bachelorarbeit.

… Herrn Dipl.-Ing. Matthias Päßler, für die ständige Diskusions- und Hilfsbereitschaft

… Herrn Prof. Dr. Georg Beier, für die fachliche Betreuung und die begleitende Unterstützung, seitens der Westsächsischen Hochschule Zwickau.

… Meiner Freundin Sandra Hain, die mich während der gesamten Bachelorzeit in jeder Hinsicht unterstützt und motiviert hat.

… Meiner Familie, die mich während des gesamten Studiums unterstützt hat.

Inhaltsverzeichnis

Abkürzungsverzeichnis

API: ..... Application Programming Interface
BCL: ..... Base Class Libary
BpP: ..... Bit per Pixel
C#: ..... C Sharp
CIL: ..... Common Intermediate Language
CLI: ..... Common - Language - Infrastructure
CLR: ..... Common Language Runtime
COM: ..... Component Object Model
GUI: ..... Graphical User Interface
Microsoft .Net CF: ..... Microsoft .Net Compact Framework
OEM: ..... Original Equipment Manufacturer
OLE: ..... Object Linking and Embedding
OPC: ..... OLE for Process Control
RGB: ..... Rot Grün Blau

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Die meisten Maschinen der Firma Mühlbauer werden über ein Graphical-User-Interface (GUI), welches auf einem Windows XP Rechner ausgeführt wird, bedient.

Für Windows XP Systeme existiert eine Steuerelementebibliothek auf Basis des Microsoft .Net Frameworks. Diese enthält Methoden für grafische Anzeige- und Steuerelemente und die Steuerelemente selbst. Zudem existiert auch ein GUI basierend auf dieser Steuerelementebibliothek.

Aus Kostengründen sollen die Windows XP Rechner zukünftig ersetzt werden. Die neu verwendeten Steuerungen basieren auf dem Betriebssystem Windows CE. Um damit auch weiterhin die direkte Bedienbarkeit zu realisieren, ist es notwendig ein GUI auf Basis des Microsoft .Net CF zu entwickeln. Für dieses neue GUI ist als erstes eine neue Steuerelementebibliothek basierend auf dem Microsoft .Net CF zu realisieren.

Als Programmierumgebung steht Microsoft Visual Studio 2005 mit dem Microsoft .Net CF 2.0 zur Verfügung, als Programmiersprache wurde C# gewählt.

1.1. Aufgabenstellung

Portierung und Revision einer Steuerelementebibliothek basierend auf dem Microsoft .Net Framework zum Microsoft .Net Compact Framework für ein GUI von Windows CE Geräten.

Die Aufgabenstellungen, mit denen sich diese Bachelorarbeit im Einzelnen befasst sind:

- Vergleich des Microsoft .Net Framework mit dem Microsoft .Net CF
- Untersuchung der aktuellen Steuerelementebibliothek
- Wichtigste Elemente auf das Microsoft .Net CF portieren und eine Steuerelementebibliothek für Windows CE entwickeln
- Verwendung der Programmiersprache C# (C Sharp)

2. Vorbetrachtung

Ein Framework ist ein Programmiergerüst, das in der Softwaretechnik, insbesondere im Rahmen der objektorientierten Softwareentwicklung, sowie bei komponentenbasierten Entwicklungsansätzen, verwendet wird.

In den folgenden Abschnitten wird eine Übersicht über das Microsoft .Net Framework und das .Net CF gegeben.

2.1. Microsoft .Net Framework und .Net Compact Framework

„Die .Net-Plattform ist die Umsetzung des Common-Language-Infrastructure-Standards (CLI) und stellt mit diesem eine Basis zur Entwicklung und Ausführung von Programmen dar, die mit unterschiedlichen Programmiersprachen auf verschiedenen Plattformen erstellt wurden. Hauptbestandteile sind die (objektorientierte) Laufzeitumgebung Common Language Runtime (CLR), die Base Class Libary (BCL) sowie diverse Hilfsprogramme.“ [1]

„Die CLR ist die Laufzeitumgebung von .Net und stellt somit den Interpreter für den standardisierten Zwischencode der Common Intermediate Language (CIL) dar. Für die CIL wurde ein sprachübergreifendes System von objektbasierten Datentypen definiert, so dass alle Hochsprachen, die sich an den CLI-Standard halten, gültigen CIL – Bytecode erstellen können.“ [1] (Siehe Abbildung 1)

Im Gegensatz zur Java – Plattform, die .Net am ähnlichsten ist, wurde .Net von Anfang an für die Programmierung mit unterschiedlichen Programmiersprachen entwickelt.

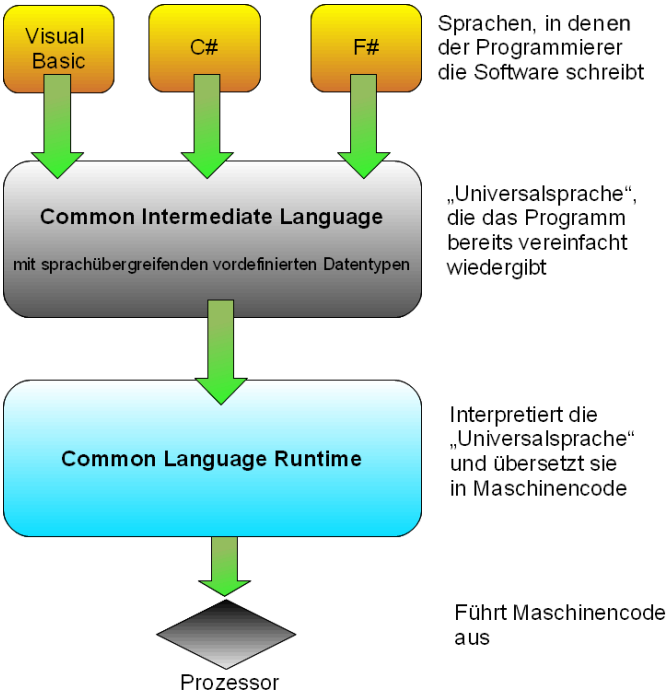

Abbildung 1 - .Net Basisprinzip

„Das .Net Framework ist ein Produkt, welches die Entwicklungsgrundlage für die Microsoft .Net Plattform bildet. Das .Net Framework und das geräteorientierte .Net Compact Framework stellen eine verwaltete, sichere Ausführungsumgebung für XML-Webdienste und Anwendungen sowie umfassende Unterstützung für XML zur Verfügung.“ [1]

Die Programmierung der .Net-Klassenbibliothek ist hundertprozentig objektorientiert. Es gibt keine Elemente die sich nicht auf Objekte zurückführen lassen. Selbst generische Datentypen (zum Beispiel Integer) werden als Objekte behandelt und Zugriffe auf das Betriebssystem werden durch Klassen gekapselt.

Da objektorientierte Programmierung bei zu wenig Sorgfalt zu Memoryleaks führt, entlastet die Einführung eines so genannten Garbage Collectors die Programmierung und führt zu höherer Sicherheit. Der Garbage Collector erkennt nicht mehr benötigte Objekte und entfernt sie automatisch aus dem Speicher.

Das .Net Compact Framework stellt eine Teilmenge des vollständigen .Net Framework dar. Es realisiert nur einen kleinen Teil der gesamten .Net Framework – Klassenbibliothek und enthält darüber hinaus spezielle Features und Klassen zur

Entwicklung für Mobile und Embedded Systeme. Dieser Teil ist für Anwendungen geeignet, die auf Geräten mit beschränkten Ressourcen zum Einsatz kommen sollen. [1] [2]

Nach den ersten Untersuchungen und Tests, die existierende Bibliothek unter .Net CF zu verwenden, hat sich gezeigt, dass die vorhandenen Steuerelemente (User Controls) sich nicht für Windows CE verwenden lassen.

2.2. Die Programmiersprache C#

„C# (C Sharp) wurde von Microsoft entwickelt und zählt zu den objektorientierten Programmiersprachen. Sie unterstützt sowohl die Entwicklung von sprachunabhängigen .Net-Komponenten als auch COM-Komponenten (Component Object Model), für den Gebrauch mit WIN32-Applikationen. Das COM ist eine Plattformtechnik, um unter dem Betriebssystem Windows Interprozesskommunikation und dynamische Objekterzeugung zu ermöglichen. C# Programme laufen dadurch auch plattformunabhängig auf allen Win32-Systemen mit installiertem Framework. Win32 ist die 32-Bit-API für moderne Versionen von Windows.“ [2]

Als .Net-Sprache verfügt auch C# über Sprachunterstützung für Attribute und Delegaten. Mit Hilfe von Attributen kann man, Informationen über eine Klasse, ein Objekt oder eine Methode speichern, die zur Laufzeit ausgewertet werden können. Delegaten in C# sind vergleichbar mit Funktionszeigern in C oder C++. Ein Delegat ist ein Objekt, welches eine Referenz auf eine Methode enthält. Delegaten haben eine Signatur, ähnlich der von Methoden. Alle Methoden, auf die ein Delegat zeigen soll müssen einer im Delegat festgelegten Signatur entsprechen. [1]

3. Übersicht über die aktuelle Steuerelementebibliothek für .Net Framework

Um den Standardsteuerelementen ein firmenspezifisches Aussehen zu verschaffen, wird jedes Steuerelement selbst gezeichnet. Dafür steht die MBRenderer Klasse zur Verfügung. Alle Steuerelemente der vorhandenen Steuerelementebibliothek, der MBControls-Bibliothek, lassen sich auf die Control- oder UserControl-Klasse des .Net Frameworks zurückführen. Diese beiden Klassen sind bei der Oberflächenerstellung die wichtigsten Elemente des .Net Frameworks.

Diese Klassen implementieren elementare Methoden, die für Klassen erforderlich sind, welche Informationen anzeigen. Sie behandeln Benutzereingaben, die über die Tastatur oder Zeigegeräte erfolgen. Zusätzlich verwalten sie die Sicherheit und das Weiterleiten von Meldungen. Sie definieren die grafischen Begrenzungen eines Steuerelements (Position und Größe), realisieren jedoch nicht das Zeichnen der spezifischen Steuerelemente. [1]

3.1. Klassendiagramme der MBControls.dll

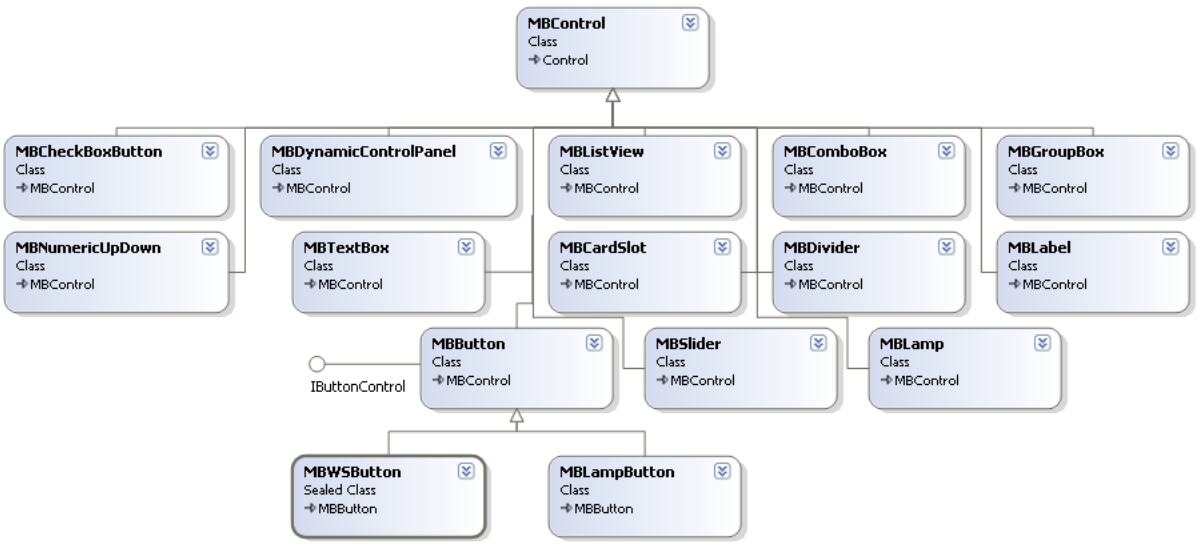

Abbildung 2 - Klassendiagramm 1 MBControls.dll

Abbildung 2 zeigt, dass eine Hauptklasse vorhanden ist, von welcher die Steuerelemente abgeleitet sind. Diese Mutterklasse, MBControl abgeleitet von der Control-Klasse des .Net Frameworks, stellt Methoden, Eigenschaften und

Schnittstellen zur Verfügung, auf welche jedes abgeleitete Steuerelement zugreifen kann. Zusätzlich enthält die MBControls-Bibliothek eine Hauptklasse für kombinierte Steuerelemente, welche Hardwarebauteile steuern können.

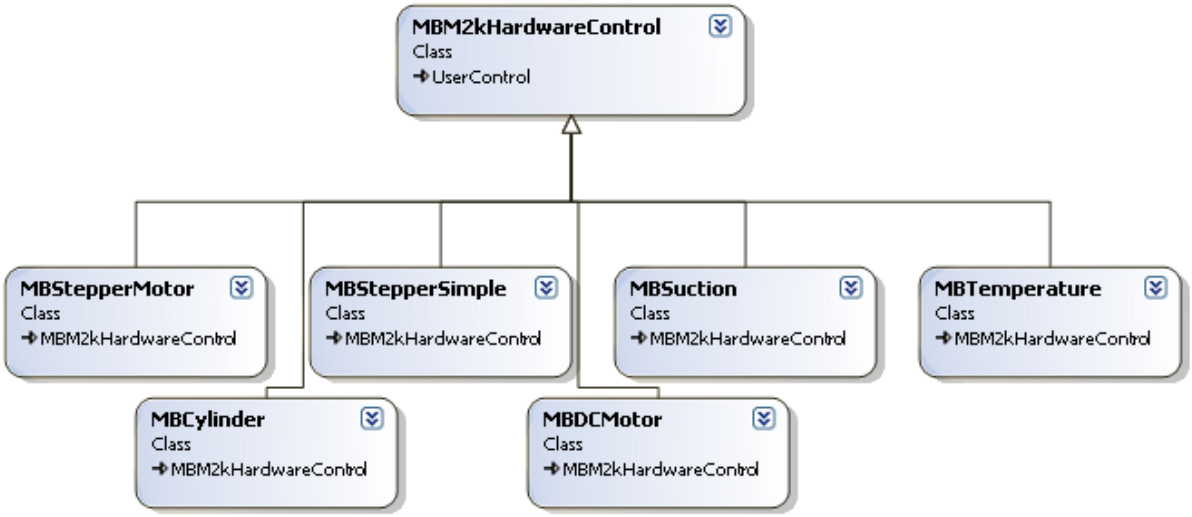

Abbildung 3 - Klassendiagramm 2 MBControls.dll

3.2. Schnittstellen der MBControls-Bibliothek

Die Steuerelementebibliothek verfügt über vier Schnittstellen:

- Eine Sprachschnittstelle
- Eine Protokollierungsschnittstelle
- Eine Benutzerrechte – Verwaltungsschnittstelle
- Eine OPC – Schnittstelle (siehe 3.2.1)

Die Sprachschnittstelle ermöglicht das Umschalten der Sprache während der Laufzeit. Das heißt, es wird für jedes Steuerelement ein Index an den Language-Service des GUI geschickt und dieser Service liefert dann den aktuellen Anzeigetext zurück.

Die Protokollierungsschnittstelle wird ebenfalls über einen GUI-Service, den Logging-Service, aufgebaut. Dieser Service stellt Methoden zur Verfügung, über welche der Programmierer Protokolleinträge erzeugen kann.

Die Benutzerrechte-Verwaltungsschnittstelle wird auch über einen Service des GUI realisiert. Dieser ermöglicht es, die Bedienung der Steuerelemente, einem bestimmten Benutzer oder einer bestimmten Benutzergruppe zuzuordnen.

3.2.1. Die OPC - Schnittstelle

OPC steht für OLE for Process Control und basiert auf dem Komponentenmodel der Firma Microsoft. Der Begriff OLE (Object Linking and Embedding) wurde von Microsoft zeitweise für die gesamte Komponentenarchitektur verwendet. OPC ist ein Standard für die Kommunikation zwischen Windows Applikationen und Geräten auf der Automatisierungsebene.

Diese Softwareschnittstelle ermöglicht den Datenaustausch zwischen Anwendungen unterschiedlichster Hersteller in der Automatisierungstechnik.

Die OPC Schnittstelle ist vollständiger Bestandteil der Software, die auf einem PC als Plattform für das GUI läuft. Für die Kommunikation über diese Schnittstelle steht im .Net Framework eine Bibliothek zur Verfügung. Da diese Bibliothek nicht für das .Net CF eingesetzt werden kann, muss diese Schnittstelle mit eigenen Klassen realisiert werden, welche schon realisiert sind und genutzt werden können.

OPC Prinzip unter Windows NT basierten Systemen

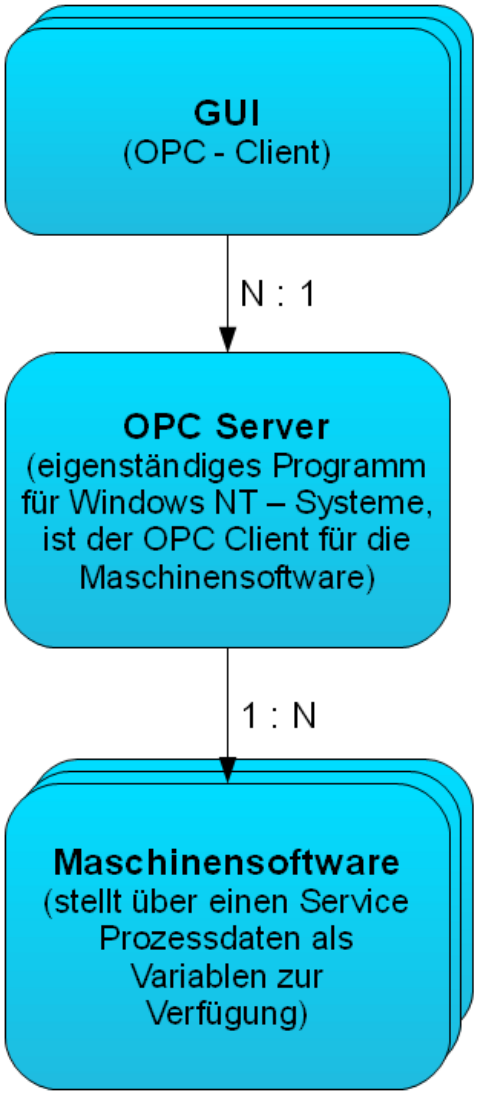

Abbildung 4 - OPC Prinzip unter Windows NT Systemen

Abbildung 4 zeigt das OPC-Server-Client-Prinzip auf Windows NT basierten Systemen. Das GUI greift als OPC Client auf den OPC Server zu, welcher als eigenständiger Prozess auf dem Windows NT Rechner läuft. Dieser wiederum greift als OPC Client auf die Maschinensoftware zu. Die Maschinensoftware stellt ihrerseits, über einen Service, Prozessdaten als Variablen zur Verfügung. Diese werden als Gruppen zusammengefasst und verschiedenen OPC Servern zur Verfügung gestellt.

OPC Prinzip unter Windows CE basierten Systemen

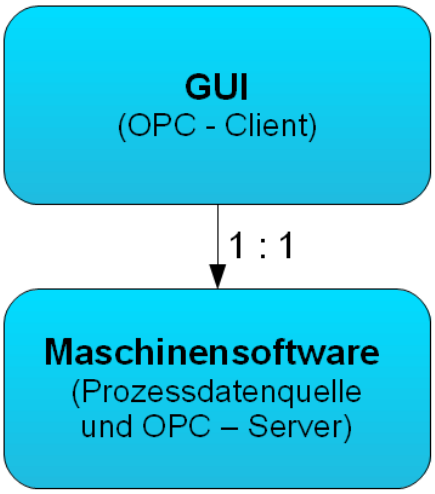

Abbildung 5 - OPC Prinzip Windows CE

Abbildung 5 zeigt das OPC-Server-Client-Prinzip auf Windows CE basierten Systemen. Hier ist der OPC Server vollständig in die Maschinensoftware integriert und das GUI verbindet sich direkt mit dem OPC Server, also der Maschinensoftware.

Allgemeines OPC Prinzip

Ein OPC Server wird von einem Hersteller, in diesem Fall von der Mühlbauer Maschinensoftware, zum Zugriff auf Prozessdaten bereitgestellt. Der Server stellt Variablen zur Verfügung, welche dann über Methoden vom Client gelesen oder geschrieben werden können.

Ein OPC Client als Nutzer der Dienste ist nicht nur auf einen Server beschränkt, sondern kann beliebig viele OPC Server nutzen. Für OPC Server werden eindeutige Namen vergeben. Diesen Namen muss der Client verwenden um einen Server zu spezifizieren.

Die Aufgabe der Maschinensoftware ist es, in festen Zeitabständen die Prozessdaten über das interne Bussystem der Hardwaresteuerung abzufragen und benötigte Variablen über den OPC Server zugänglich zu machen. Der OPC Client kann dann auf die Änderungen reagieren oder selbst die Variablen ändern.

Die OPC Spezifikation für Data-Access teilt die Schnittstelle und deren Methoden in drei hierarchische Klassen ein, das Klassenmodell ist in Abbildung 6 dargestellt und wird in den folgenden Abschnitten beschrieben.

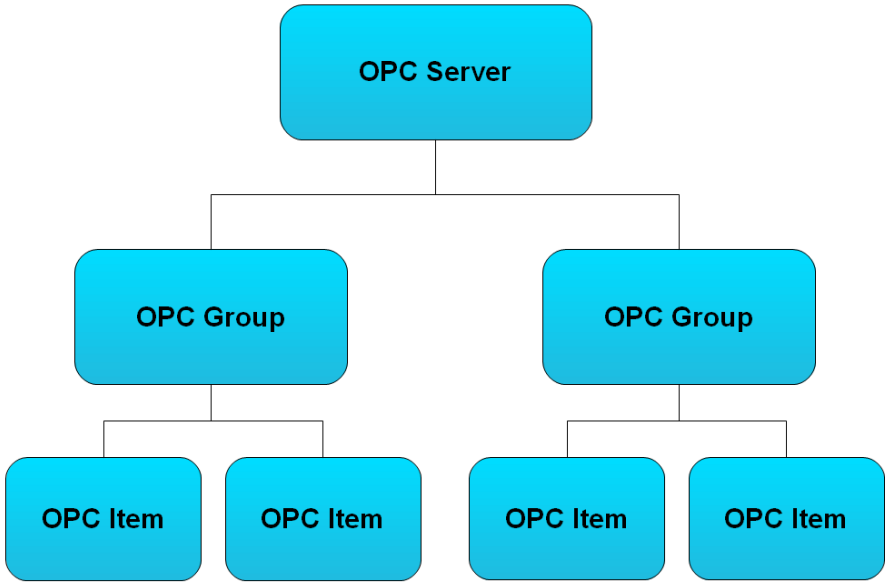

Abbildung 6 - OPC – Klassenmodell

OPC Server

An oberster Stelle steht die Klasse OPC Server. Ein Objekt dieser Klasse repräsentiert einen spezifischen OPC Server. Diese Klasse besitzt verschiedene Attribute und Methoden, die Informationen über den Status, die Version und den Adressraum der verfügbaren Prozessvariablen zu liefern.

OPC Group

Die Klasse OPC Group strukturiert die vom OPC Server genutzten Prozessvariablen. Mit Hilfe der Objekte dieser Klasse kann ein OPC Client Mengen von Prozessvariablen bilden.

OPC Item

Ein Objekt dieser Klasse repräsentiert eine Verbindung zu einer Prozessvariablen. Eine Prozessvariable ist ein Element des Adressraumes des OPC-Servers. Identifiziert wird ein OPC Item durch dessen ItemID, welche in den Steuerelementen als OPC Path bezeichnet wird. Die ItemID wird vom Hersteller des Servers, also der

Maschinensoftware, festgelegt. Mit jedem OPC Item sind die Eigenschaften Wert, Qualität und Zeitstempel verbunden. Die Qualität sagt aus, ob der Wert der Variablen sicher ermittelt werden konnte und bestimmt somit die Aussagekraft des Wertes.

Um den, durch die Abfragezyklen (Polling) und Callbacks verursachten Datenverkehr möglichst gering zu halten, beinhalten die Gruppen die Möglichkeit, die Aktualisierung zu unterbinden, indem man entsprechende Gruppen sperrt bzw. freigibt. So kann man beispielsweise nur die Zustände von Sensoren aktuell halten, die auch gerade angezeigt werden. [3]

4. Entwicklung der Steuerelemente Bibliothek für .Net CF

4.1. Vorgehensweise

Schrittweise werden die einzelnen GUI-Elemente analysiert, portiert und getestet.

4.1.1. Grafische Analyse der .Net Framework Bibliothek

Es wird ein minimales GUI für das .Net Framework entwickelt, das immer nur das zu testende Steuerelement enthält. Auch ein GUI für das .Net CF wurde entwickelt um die neuen Steuerelemente zu testen. Als erstes wird die grafische Darstellung des vorhandenen Elementes während der Laufzeit untersucht. Ein neues Steuerelement basierend auf dem .Net CF wird erstellt und der vorhandene Code für die Darstellung übernommen. Dabei werden voraussichtlich Fehler auftreten wegen dem Fehlen von Zeichenmethoden und Zeichenattribute im .Net CF. Daraus ergibt sich eine Neuentwicklung des Codes für die Darstellung. Testen dieser Darstellung des neuen Steuerelementes mit dem entsprechenden GUI und der entsprechenden Hardware.

4.1.2. Funktionsanalyse der .Net Framework Bibliothek

Das vorhandene Steuerelement wird hinsichtlich seiner Funktionalität während der Laufzeit, mit Hilfe des minimalen GUI, überprüft und die Implementierung der einzelnen Methoden wird untersucht. Die vorhanden Methoden werden im .Net CF

nach gebildet und teilweise verbessert. Die Funktionalität des neuen Steuerelementes wird mit Hilfe des GUI für .Net CF getestet.

4.2. Allgemeines

Für das firmenspezifische Aussehen der Steuerelemente wurde in der .Net Framework Bibliothek eine extra MBRenderer Klasse erzeugt, in der Methoden für das Zeichen der Steuerelemente implementiert sind. Eine solche Klasse wird auch für das .Net CF erstellt und erhält nach und nach immer weitere Methoden für das Zeichnen der Steuerelemente. Auf selbige wird im weiteren Verlauf dieser Bachelorarbeit immer wieder eingegangen. Jedes Steuerelement besitzt eine OnPaint Methode. Durch das Überladen dieser Methode kann man jedem Steuerelement ein individuelles Aussehen ermöglichen.

4.3. Portierung der Schnittstellen

Die vorhandenen Schnittstellen, der MBControl Bibliothek, können unverändert übernommen werden, da die eigentlichen Methoden dafür von dem GUI bereitgestellt werden sollen.

Für die .Net CF Bibliothek wurde zusätzlich eine neue Schnittstelle hinzugefügt. Die Kurzwahl (Hotkey) – Schnittstelle, mithilfe dieser kann man bestimmten Tasten verschiedene Methoden zuweisen.

4.4. Die Basisklasse MBControl_CE

Da fast alle Elemente von dieser Klasse erben, kann man hier die Hauptfunktionalitäten der Steuerelemente unterbringen.

Diese Klasse ist von der Control Klasse aus dem .Net CF abgeleitet, welche schon standardmäßig einige Eigenschaften und Methoden zu Verfügung stellt. MBControl_CE enthält die Haupteigenschaften für fast alle Steuerelemente. Um eine

Eigenschaft zu verändern wird ein privates Attribut der Klasse mittels einer Get- und Set-Methode öffentlich zugänglich gemacht.

Zu diesen Eigenschaften gehört unter anderem ein OPC Item, welches dann in den abgeleiteten Steuerelementen abgefragt oder verändert werden kann. Zu diesem OPC Item gehört weiterhin eine virtuelle Aktualisierungsfunktion. Diese kann im jeweiligen Steuerelement überschrieben werden. Damit kann auf Ereignisse, wie zum Beispiel eine Sensoränderung reagiert werden, um diese dann im Steuerelement darzustellen oder in eine Variable zu speichern. Zu dem OPC Item gehört auch ein OPC Path, welcher die ItemID Eigenschaft darstellt. Es werden in dieser Basisklasse zusätzlich zwei Methoden realisiert, welche das Runden und Abschneiden von Nachkommastellen für numerische Zeichenketten ermöglichen. Somit hat der Programmierer die Möglichkeit numerische Zeichenketten vor der Darstellung in Steuerelementen automatisch ändern zu lassen.

Zusätzlich wird die TextToDraw Eigenschaft festgelegt. Diese Eigenschaft setzt sich aus insgesamt drei Zeichenketten zusammen und kann, zum Beispiel zur Darstellung eines Textes, auf dem jeweiligen abgeleiteten Steuerelement genutzt werden. Die Zeichenkette setzt sich, aus einem TextPrefix, welcher vor dem darzustellenden Text steht, dem darzustellenden Text selbst und einem TextPostfix, welcher hinter dem Text steht zusammen. Der darzustellende Text selbst kann dabei entweder über die Sprachschnittstelle, falls diese verfügbar ist, über eine Eigenschaft des Steuerelementes oder über das OPC Item gesetzt werden.

4.5. Allgemeines zur Grafische Darstellung im .Net CF

Einige Steuerelemente aus der MBControl.dll werden mit einem Farbverlauf (Gradient) gezeichnet. Im .Net Framework existieren Methoden, welche die Darstellung eines Gradienten ermöglichen. Abbildung 7 zeigt einen solchen Farbverlauf.

Abbildung 7 - Beispiel Farbverlauf (Gradient)

Im .Net CF existieren diese Methoden nicht, deshalb ist es notwendig diese selbst zu entwickeln. Es wurden 2 verschiedene Routinen untersucht.

- Gradienten Darstellung mit Hilfe der Coredll.dll (einer Windows CE System Bibliothek)
- Gradienten Darstellung mit Hilfe von vorgefertigten Bilddateien

4.5.1. Gradienten Darstellung mit Hilfe der Coredll.dll

Für diese Routine ist es notwendig eine Kommunikationsklasse zu schreiben, welche auf die GradientFill – Methode aus der Windows CE spezifischen Coredll.dll zugreift.

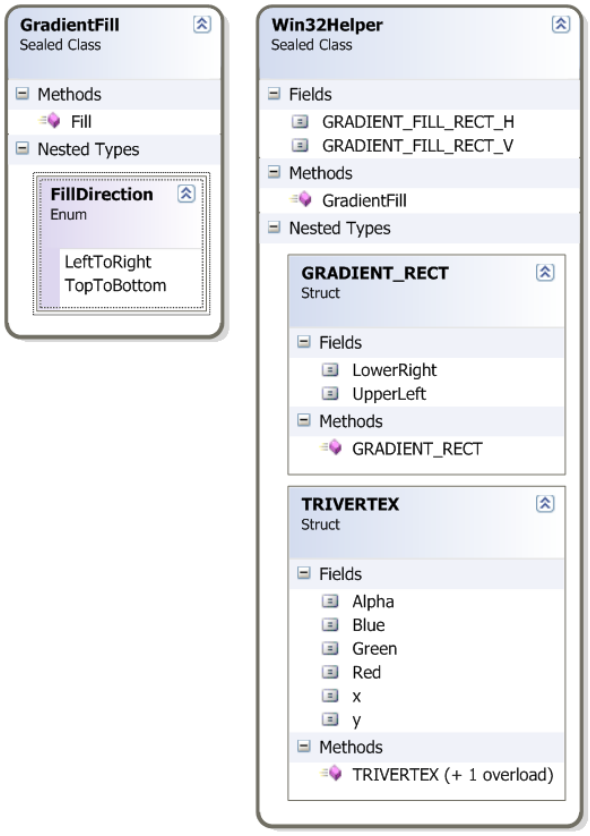

Abbildung 8 - Klassendiagramm GradientFill

Wie in Abbildung 8 zu sehen ist, ist in der Win32Helper Klasse der Zugriff auf die GradientFill Methode aus der Coredll Bibliothek implementiert. Für den Programmierer wurde zudem eine GradientFill Klasse entwickelt, welche die Methode Fill beinhaltet. Mit dieser Methode kann der Entwickler nun einfach ein beliebiges Rechteck, mit einem selbst definierten Gradienten zeichnen. Beim

entwickeln eines Buttons mit dieser Art der Darstellung, stellte sich folgendes Problem heraus.

Abbildung 9 - Button mit Coredll.dll

Wie Abbildung 9 zeigt kann der Button während der Oberflächengestaltung, mit einem visuellen Designer nicht richtig dargestellt werden. Dieser kann nur während der Laufzeit, mit der Windows CE spezifischen Coredll.dll gezeichnet werden. Dadurch kann der Entwickler während der Programmentwicklungsphase nicht sehen, ob er den Button richtig konfiguriert hat, aus diesem Grund wurde noch eine andere Methode für die Darstellung von Gradienten untersucht.

4.5.2. Gradienten Darstellung mit Hilfe von vorgefertigten Bilddateien

Um Steuerelemente mit Gradient nun auch während der Oberflächenentwicklung richtig darzustellen, wurde auf die Darstellung mit Hilfe von Bitmaps zurückgegriffen. Das heißt, es wurde mit einem Zeichenprogramm ein Bild erzeugt, welches das gewünschte Gradientenmuster enthält und dieses Bild wurde dann auf das gewünschte Steuerelement gezeichnet.

Der Nachteil besteht in der Abhängigkeit von den vorgefertigten Bildern. Eine Farbanpassung ist nicht so schnell möglich und immer mit Programmieraufwand verbunden. Aufgrund des vorgegebenen Designs, kommt dieser Nachteil aber kaum zum Tragen.

Die Größe des Bildes kann beliebig gewählt werden, da das .Net CF Zeichenfunktionen enthält, mit welchen man Bilder vergrößern oder verkleinern kann. Durch diese Verfahren ist die Darstellung während der Oberflächengestaltung möglich und der Programmierer kann die Steuerelemente richtig konfigurieren. Abbildung 10 zweigt einen Button, welcher mit einem Bild dargestellt wurde.

Abbildung 10 - Button mit Bild

5. Einfache Steuerelemente

5.1. Der Mühlbauer Button (MBButton)

Dieses Bedienelement stellt das Grundelement vieler anderer Steuerelemente dar. Für die Darstellung des Buttons wurden für jede gewünschte Farbe zwei Bitmaps mit den jeweiligen Gradienten gezeichnet. Dadurch kann man zur Entwicklungszeit oder während der Laufzeit entscheiden, welche Farbe und welchen Zustand der Button haben soll. Es existieren im .Net Framework einige Mausbehandlungsroutinen, auf die der Mühlbauer Button aus der MBControl-Bibliothek zugreift. Diese existieren im .Net CF nicht. Beispiele sind das MouseEnter-, das MouseLeave- und das MouseHover-Event.

Die Nutzung dieser drei Methoden ermöglicht es, dem originalen Button vier verschiedenen Stati darzustellen. Einen, wenn der Button gedrückt ist, einen, wenn sich der Mauszeiger über ihm befindet, einen Normalen, wenn nichts passiert und einen, wenn der Button gesperrt ist. Das MouseHover-Event ist im .Net CF nicht vorhanden, dadurch entfällt hier der Status, wenn sich die Maus über dem Button befindet. Für jeden der verbleibenden Stati wurden eigene Bilder erzeugt, wie Abbildung 11 zeigt.

Abbildung 11 - Die verschiedenen Button Stati (normal, gedrückt und gesperrt)

Die Darstellung des Buttons wird in die MBRenderer Klasse ausgelagert, das heißt, in der überladenen OnPaint-Methode des Buttons wird die RenderButton-Methode aus der MBRenderer Klasse aufgerufen.

```
public static void RenderButton(Graphics graphics, Rectangle bounds, ButtonStates state,
ButtonColors color)
{
    SetColor(color);
    Bitmap buttonBitmap;
    switch (state)
    {
        case ButtonStates.Disabled:
            buttonBitmap = ButtonDisabledBitmap;
            break;
        case ButtonStates.Pressed:
            buttonBitmap = ButtonPressedBitmap;
            break;
        case ButtonStates.Normal:
        default:
            buttonBitmap = ButtonNormalBitmap;
            break;
    }
    RoundedRectangle rr = new RoundedRectangle(bounds, 8);
    Rectangle destinationRect = new Rectangle(bounds.X + 2, bounds.Y + 2,
        bounds.Width - 4, bounds.Height - 4);
    Rectangle sourceRect = new Rectangle(m_LeftMargin, m_TopMargin,
        buttonBitmap.Width - (m_LeftMargin + m_RightMargin),
        buttonBitmap.Height - (m_BottomMargin + m_TopMargin));
    graphics.DrawImage(buttonBitmap, destinationRect, sourceRect, GraphicsUnit.Pixel);
    graphics.DrawPolygon(new Pen(Color.DimGray, 2), rr.GetButtonPoints());
    return buttonBitmap;
}
```

Abbildung 12 - RenderButton - Methode aus der MBRenderer Klasse

Abbildung 12 zeigt eine kurze Übersicht über die RenderButton-Methode, welche im Folgenden erläutert wird. An diese Methode wird ein Graphics Objekt, ein Rechteck mit der Größe des Buttons, der aktuelle Status und die gewünschte Buttonfarbe übergeben.

Über den Eingangsparameter color wird die aktuell darzustellende Farbe des Buttons gesetzt. Diese wird dann in der SetColor-Methode ausgewertet und das jeweilige Bild passend zu der ausgewählten Farbe auf zwei Attribute der MBRenderer Klasse geschrieben. Danach wird eine Bitmap erstellt, und je nach dem mit übergebenen Buttonstatus das jeweilige Attribut auf diese Bitmap gespeichert.

Wie bereits im Vorfeld erwähnt, sollen sich die Mühlbauer Steuerelemente von den Standard Windows Elementen abheben. Im Falle des Buttons werden abgerundete Ecken gezeichnet, dafür wurde die Klasse RoundedRectangle erstellt, welche in den folgenden Absätzen kurz erläutert wird.

Die RoundedRectangle Klasse besteht aus zwei Attributen, einem Rectangle (Rechteck – eine Struktur im .Net CF) und einer Integer Zahl für den Radius der Ecken. Im Konstruktor werden beide Attribute gesetzt, um dann in der Methode GetButtonPoints verwendet zu werden.

Im .Net Framework existiert eine Klasse GraphicsPath. Die RoundedRectangle Klasse in der MBControl.dll hat ein Attribut von dieser Klasse. Diesem Attribut werden hier Linien und Winkel hinzugefügt, um ein abgerundetes Rechteck entstehen zu lassen. Dieses GraphicsPath Objekt kann man dann im weiteren Code zeichnen lassen.

Das .Net CF beinhaltet keine Klasse GraphicsPath und keine Winkelmethode. Eine Rundung ist nur durch einen Polygonzug darstellbar, wozu man ein Punktefeld benötigt. Dieses Feld wird so gefüllt, dass ein Rechteck mit abgerundeten Ecken entsteht. Das so entstandene Feld, wird dann in einer Zeichenroutine des .Net CF automatisch zu einer Linie zusammengefasst.

Das endgültige Zeichnen erfolgt dann mittels zwei von .Net CF bereitgestellten Methoden. DrawImage, welche das ausgewählte Bild des Buttons zeichnet und DrawPologon, welche von dem erzeugten RoundedRectangle Objekt das Punktefeld zeichnet. Damit der Button auch sichtbar mit runden Ecken gezeichnet wird, muss man die Eigenschaft BackColor noch auf die Farbe des aktuellen Hintergrundes setzen.

Beim Belegen des Buttons mit einem Text, kann dieser mit folgenden Eigenschaften parametrisiert werden.

- Die Ausrichtung des Textes auf dem Button
- Der Texteinzug des Textes
- Die Schriftart
- Die Schriftfarbe

Als Text wird das TextToDraw Attribut aus der MBControl_CE Klasse verwendet. Für die Ausrichtung des Textes wurde eine Enumeration angelegt, welche die Ausrichtungen enthält. Zur Auswahl stehen:

- TopLeft
- TopCenter

- TopRight
- MiddleLeft
- MiddleCenter
- MiddleRight
- BottomLeft
- BottomCenter
- BottomRight

Die Ausrichtung kann über die TextAlign Eigenschaft des Buttons gewählt werden. In der dazugehörigen MBRenderer-Methode wird dann die Posittion des Textes berechnet.

Die Schriftart und die Farbe stellt man über die Standardeigenschaften eines Steuerelementes ein. Für den Texteinzug wird ebenfalls eine extra Eigenschaft bereitgestellt. Alle diese Werte und zusätzlich noch das Rechteck des Buttons werden dann wieder an die Methode, RenderText aus der MBRenderer-Klasse übergeben. Diese Methode berechnet dann die Position des Textes und zeichnet selbigen.

Zusätzlich zu dem Text gibt es noch die Möglichkeit ein Bild (Icon) auf dem Button darzustellen. Ähnlich wie bei dem Text, werden hierfür noch weitere Eigenschaften zu dem Button hinzugefügt:

- eine Bilddatei
- die Ausrichtung für das Bild
- der Bildeinzug
- die Skalierung des Bildes (in Prozent).

Die Bilder für den Button sollen den Nutzer später unterstützen, die Funktionsweise des Buttons zu verstehen, wie Abbildung 13 zeigt.

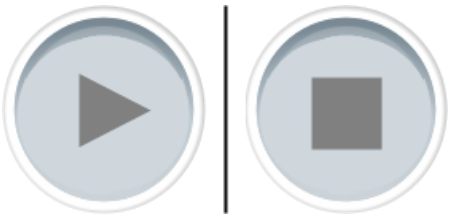

Abbildung 13 - Button Bilder (Start, Stopp)

Diese Bilder existieren für die MBControl Bibliothek und sollen ebenso für .Net CF genutzt werden. Der Hintergrund dieser Bilder ist transparent. Bei der Darstellung dieser Bilder auf dem neuen Button hat sich das Problem gezeigt, dass das Betriebssystem Windows CE keine Transparenz unterstützt, wie in Abbildung 14 dargestellt wird.

Abbildung 14 - Versuch Transparentes Zeichnen

Jedem Pixel, der in dem Button Bild als Transparent definiert ist, wird unter Windows CE eine Farbe zugeordnet. Das heißt, die Bilder konnten nicht in dieser Art dargestellt werden. Es musste ein Weg gefunden werden um zwei unterschiedliche Bilder aufeinander darzustellen, wobei eines der beiden Bilder Transparenz enthalten soll.

Die Idee ist, den transparenten Pixeln des Bildes einen bestimmten Farbwert zuzuordnen, dann die beiden Bilder übereinander zu zeichnen aber den definierten Farbwert durch den aktuellen Farbwert des Hintergrundes zu ersetzen, so dass das resultierende Bild eine Quasitransparenz aufweist.

Abbildung 15 - Transparenz und Quasitransparenz

Abbildung 15 zeigt links das originale Bild mit echter Transparenz und rechts das veränderte Bild mit Grün als transparenter Farbe. Grün wurde gewählt, da es aufgrund der Farbvorgaben des existierenden GUI unwahrscheinlich ist, dass diese Farbe zum Einsatz kommt.

Um diesen Ansatz zu realisieren wurden drei verschiedene Varianten untersucht. Durch interne Zeitmessungen bei Versuchen wurde festgestellt, dass dieser Vorgang mit einigen Varianten sehr zeitaufwendig ist.

Erste Variante – GetPixel und SetPixel aus der Bitmap Klasse

```
public Bitmap MergeWithGetAndSetPixel(Bitmap background,Bitmap foreground)
{
    Bitmap merged = background;
    Color transparent_color = Color.FromArgb(0, 255, 0);
    for (int x = 0; x < foreground.Width; x++)
    {
        for (int y = 0; y < foreground.Height; y++)
        {
            if (foreground.GetPixel(x, y) != transparent_color)
            {
                merged.SetPixel(x, y, foreground.GetPixel(x, y));
            }
        }
    }
    return merged;
}
```

Abbildung 16 - Code für GetPixel und SetPixel

Diese Methode arbeitet mit den Bitmap Operationen, SetPixel und GetPixel, welche vom .Net CF bereitgestellt werden. Als Eingabeparameter erhält die Methode zwei Bilder. Das eine Bild ist der gewünschte Hintergrund und das andere ist das Quasitransparente Bild, welches auf dem Hintergrund dargestellt werden soll. Bei diesem Bild ist schon die Transparenz mit Grün ersetzt worden. Nacheinander wird von jedem Pixel des Bildes der Farbwert, mit dem Farbwert der quasitransparenten Farbe verglichen. Falls diese nicht übereinstimmen, das heißt, falls der Pixel nicht transparent sein soll, wird er auf das Hintergrundbild gezeichnet. Das so entstandene Bild wird dann von der Methode zurückgegeben und kann daraufhin dargestellt werden.

Zweite Variante – Transparenz mit der LockBits – Methode

Die Bitmap Klasse des .Net CF stellt des weiteren die LockBits- und die dazugehörende UnlockBits-Methode zur Verfügung, welche es ermöglichen einen Teil der Pixeldaten des Bildes in den Speicher zuladen und zu manipulieren. Die LockBits-Methode liefert ein Objekt der BitmapData Klasse zurück, welche das Layout und die Positionen der Daten in einem Feld beschreibt.

BitmapData gibt die Attribute der Bitmap an: Pixelformat, die Anfangsadresse der Pixeldaten im Speicher (Scan0) und die Länge der einzelnen Scanzeilen (Stride). [1]

Um nun jeden Pixelwert des quasitransparenten Bildes zu untersuchen, muss man, wie bei der ersten Methode, jeden Pixel mittels zwei Schleifen adressieren. Damit der richtige Farbwert der Pixel berechnet werden kann, ist es wichtig zu wissen, welches Pixelformat das Bild hat. Das folgende Beispiel zeigt, wie man Zugriff auf ein bestimmtes Pixel mit dem jeweiligen Pixelformat hat.

- **32 Bits per Pixel (BpP)** Die Adresse des ersten Elementes des Pixel ist (y * Stride) + (x * 4). Dieser Wert zeigt auf das Blaue Byte des Pixels, die folgenden drei Bytes enthalten den Grün, Rot und Alpha Wert des Pixels.
- **24 BpP**: Die Adresse des ersten Elemente des Pixel ist (y * Stride) + (x * 3). Dieser Wert zeigt auf das Blaue Byte des Pixels und die folgenden zwei Byte enthalten den Grün und Rot Wert des Pixels.
- **8 BpP**: Die Adresse des Bytes ist (y * Stride) + x.

Im behandelten Fall haben die Bilder alle ein Pixelformat 32 BpP. Daraus ergibt sich folgender Code (Abbildung 17) zum überprüfen des Farbwertes.

```
for (int x = 0; x < fore.Width; ++x)
{
    for (int y = 0; y < fore.Height; ++y)
    {
        rgb_forecolor = y * pbits_foreground.Stride + 4 * x;
        rgb_backcolor = y * pbits_background.Stride + 4 * x;
        if (rgbValues_foreground[rgb_forecolor + 2] == refcolor.R &&
            rgbValues_foreground[rgb_forecolor + 1] == refcolor.G &&
            rgbValues_foreground[rgb_forecolor + 0] == refcolor.B)
        {
            rgbValues_foreground[rgb_forecolor+2] =
                    rgbValues_background[rgb_backcolor+2];
            rgbValues_foreground[rgb_forecolor+1] =
                    rgbValues_background[rgb_backcolor+1];
            rgbValues_foreground[rgb_forecolor+0] =
                    rgbValues_background[rgb_backcolor+0];
        }
    }
}
```

Abbildung 17 - Codeauszug für LockBits

Rgb_forecolor und rgb_backcolor ist jeweils der erste Farbwert des Pixels als Integer Zahl. rgbValues_foreground und rgbValues_background ist jeweils ein Bytefeld,

welches die RGB-Werte des jeweiligen Bildes enthält. Der aktuelle Pixel des Vordergrundbildes wird nun mit der Referenzfarbe, also der quasitransparenten Farbe verglichen und falls diese übereinstimmen, wird der Pixel auf die Farbe des dazugehörigen Hintergrundpixels gesetzt. Am Ende dieser Methode muss man noch die UnlockBits-Methode der beiden Bilder rufen, um somit die Pixeldaten wieder freizugeben. Im Anschluss daran kann das fertige Bild dargestellt werden.

Dritte Variante – Nutzung von ImageAttributes

Ein ImageAttributes Objekt verwaltet mehrere Farbanpassungseinstellungen:

- Farbanpassungsmatrizen
- Anpassungsmatrizen für Graustufenwerte
- Gammakorrekturwerte
- Farbzuordnungstabellen

  und
- Farbschwellenwerte.

Die Farben können dadurch während des Zeichnens korrigiert, abgedunkelt, aufgehellt oder entfernt werden. In dem Fall, des quasitransparente Zeichnens von Bildern, wird die SetColorKey-Methode aus dem ImageAttributes Objekt genutzt.

Diese Methode legt die hohen und die niedrigen Colorkeywerte fest, sodass ein Bereich von Farben transparent angezeigt werden kann. Jede Farbe, deren Werte für ihre drei Komponenten (Rot, Grün, Blau), jeweils zwischen den entsprechenden Komponenten der hohen und niedrigen Colorkeys liegen, wird transparent angezeigt. [1]

```
private Bitmap MergeWithImageAttributes(Bitmap background, Bitmap foreground)
{
    Bitmap merged = background;
    Graphics mergedgraphics = Graphics.FromImage(merged);
    Color transparet_color = Color.FromArgb(0, 255, 0);
    ImageAttributes ia = new ImageAttributes();
    ia.SetColorKey(transparet_color, transparet_color);
    Rectangle destination = new Rectangle(0, 0, foreground.Width,
                                foreground.Height);
    mergedgraphics.DrawImage(foreground, destination, 0, 0, foreground.Width,
                                foreground.Height, GraphicsUnit.Pixel, ia);
    return merged;
}
```

Abbildung 18 - Code für ImageAttributes

Die hohen und niedrigen Colorkeywerte werden jeweils auf die als Transparent definierte Farbe gesetzt, wie Abbildung 18 zeigt. Dann wird das Vordergrundbild mit diesen Attributen auf das Hintergrundbild gezeichnet, wodurch eine Quasitransparenz entsteht.

Zeitmessungen der drei Varianten

Für die Zeitmessungen wurde eine kleine GUI für Windows CE und das .Net CF geschrieben. In diesem GUI werden diese drei vorgestellten Varianten, in je einer eigenen Methode implementiert. Diese werden jeweils gleich oft, innerhalb einer Schleife aufgerufen und das Bild, welches man als Rückgabewert dieser Methoden erhält, wird gezeichnet. Vor Beginn jeder Schleife und nach Beendigung selbiger, wurde ein eindeutiger Zeitstempel gespeichert. Dieser Zeitstempel ist der DateTime.Now.Ticks Wert, wobei ein Tick 100 Nanosekunden entspricht.

Um eindeutige Aussagen über das Zeitverhalten jeder einzelnen Variante zu treffen, wurde jede Methode mit den gleichen Eingangsparametern aufgerufen. In der folgenden Tabelle werden die Messwerte vorgestellt.

	1 mal	**10 mal**	**50 mal**	**100 mal**	**1000 mal**
Variante 1	3000 ms	34000 ms	174000 ms	343000 ms	4120000 ms
Variante 2	< 1 ms	2000 ms	10000 ms	20000 ms	275000 ms
Variante 3	< 1 ms	20 ms	1000 ms	2000 ms	21000 ms

Tabelle 1 - Zeitmessung der Zeichenroutinen

Tabelle 1 zeigt, wie oft die jeweiligen Methoden aufgerufen wurden und wie lange dieser Vorgang gedauert hat.

Die Anzahl der Methodenaufrufe lässt sich auf die Anzahl der darzustellenden Buttons, in dem zukünftigen GUI projektieren. Das heißt, dass man sich im Bereich zwischen 50 und 100 Button bewegen wird, welche gleichzeitig dargestellt werden müssen.

Variante 1 und Variante 2 sind für die Anzahl der vorrausichtlich darzustellenden Buttons zu langsam, wodurch auch das GUI sehr langsam und träge werden würde.

Somit wird die dritte und schnellste Variante gewählt, um die quasitransparente Darstellung zu implementieren.

Mit dieser Variante wird nun das Buttonbild auf den Button gezeichnet. Über den Skalierungsfaktor kann man die Größe des Bildes einstellen. Abbildung 20 zeigt nun den komplett konfigurierten Button.

Abbildung 19 - Kompletter Mühlbauer Button

Um die Größe des Buttons dynamisch zu gestallten, das heißt, dass sich die Breite des Buttons dem Text anpasst, wird noch eine AdjustSize-Methode bereitgestellt. In dieser wird die Größe des darzustellenden Textes, anhand seiner Länge und Schriftart, berechnet und die Breite des Buttons automatisch angepasst. Um Einzustellen, ob sich die Größe des Buttons automatisch ändern soll, gibt es das boolesche Attribut AutoFitToTextWidth.

Einbinden des OPC Items aus der MBControl_CE Klasse

Um das OPC Item richtig zu parametrisieren werden noch zusätzliche Attribute zu dem Button hinzugefügt. Der Pfad des OPC Items (also die Item-ID) und drei boolesche Variablen zum Einstellen von OPC spezifischen Attributen, AutoOPCHandling, DoOPCRefresh und DoOPCRefrashOnlyOnItemChange. Um dieses Item später mit dem OPC Server richtig zu verbinden, existiert in der Mutterklasse eine Methode, AddToOPCGroup. Mit dieser Methode wird das Item in eine OPC Gruppe hinzugefügt. Falls diese Gruppe mit der OPC Server verbunden ist, wird auch das Item automatisch verbunden.

Der Mühlbauer Button verfügt zusätzlich über die Eigenschaft Pressed. Mit dieser kann man den Status des Buttons lesen und setzen. Ist das OPC Item des Buttons

mit dem OPC Server verbunden, so kann dieser Status auch über die Maschinensoftware gesetzt werden und über den Click-Callback des Buttons wird dieses Item mit wahr oder falsch beschrieben. Um den Status des Buttons über das OPC Item zu aktualisieren steht ein virtueller Callback aus der MBControl_CE-Klasse zur Verfügung. Dieser wird im Button und den anderen Steuerelementeklassen, die den Callback benötigen, überladen. Unter .Net CF ist es so, dass die grafische Darstellung und die OPC Implementierung bzw. die OPC Aktualisierung in verschieden Threads ablaufen. Dadurch ist es erforderlich, falls über die OPC-Schnittstelle einer Veränderung registriert wird, den Button-Thread mit dem OPC-Thread zu synchronisieren. (siehe Abbildung 20)

```
protected override void OPCHandler(object value, MBOPC.OPCItemQualities quality)
{
    object sender = System.Threading.Thread.CurrentThread;
    OpcShowRefreshArgs e = new OpcShowRefreshArgs(value, quality);
    OpcShowRefresh(sender, e);
}

private void OpcShowRefresh(object sender, OPC.Data.OpcShowRefreshArgs e)
{
    // Make sure we're on the right thread
    if (this.InvokeRequired == false)
    {
        this.Pressed = Convert.ToBoolean(e.value);
    }
    // Transfer control to correct thread
    else
    {
        OpcShowRefreshHandler OpcShowRefreshHandler = new
              OpcShowRefreshHandler(OpcShowRefresh);
        IAsyncResult res = BeginInvoke(OpcShowRefreshHandler,
              new object[] { sender, e });
    }
}
```

Abbildung 20 - OPC Integration in den Button

In dem OPCHandler-Callback wird ein Objekt angelegt, welches den aktuellen Thread enthält. Und um die Eingangsparameter dieser Methode weiter zu geben, wird ein OPCShowRefreshArgs Objekt mit diesen Parametern angelegt. Diese beiden Objekte werden dann an die OPCShowRefresh Methode weitergegeben. In dieser Methode wird überprüft ob der Button sich in dem aktuellen Thread befindet oder ob er synchronisiert werden muss. Falls eine Synchronisation notwendig ist wird diese mit Hilfe der BeginInvoke Methode erzeugt. Das heißt die beiden Threads werden synchronisiert und dann die OPCShowRefresh Methode erneut ausgeführt. Nun kann auf die Änderung der OPC Item reagiert werden.

5.2. Das MBLabel

Die Implementierung des MBLabels für .Net CF konnte fast unverändert von dem MBLabel für .Net Framework übernommen werden, deshalb wird das Steuerelement hier nur kurz erläutert. Es musste nur das OPC Aktualisierungsverhalten, wegen der Threadsynchronisation, wie bei dem MBButton angepasst werden.

Bei dem MBLabel handelt sich um ein Steuerelement, welches Text darstellen kann. Entweder wird der Text fest definiert, dynamisch aus der Sprach-Schnittstelle generiert oder er wird über das OPC Item des MBLabels gesetzt. Das Label bekommt noch die zusätzlichen Eigenschaften DecimalPlaces und RoundingMethod. Damit kann der Entwickler bestimmen wie der Anzeigewert dargestellt werden soll. Das ist zum Beispiel wichtig wenn der Text des MBLabels eine Gleitkomma Zahl ist und über das OPC Item gesetzt wird. Man kann entscheiden wie viel Nachkommastellen dargestellt werden und ob diese gerundet oder einfach abgeschnitten werden. Die Methoden für das Runden und Abschneiden werden in der MBControl_CE-Klasse bereitgestellt. Im Label wird dann das Text Attribut überladen. In der überladenen Set-Methode des Attributes wird dann geprüft, welche RoundingMethod ausgewählt wurde und je nachdem der anzuzeigende Text bearbeitet. Abbildung 21 zeigt das MBLabel.

Abbildung 21 - Mühlbauer Label

5.3. Die MBTextbox

Ähnlich wie bei dem MBLabel konnte die MBTextbox aus der Bibliothek für das .Net Framework übernommen werden, es musste nur das OPC Aktualisierungsverhalten, wie bei dem MBButton angepasst werden.

Dieses Steuerelement besteht aus einer Standard Textbox. Allerdings wurde hier ebenfalls die OnPaint Methode überladen um ein eigenes Aussehen zu ermöglichen. Die Standard TextBox wird ohne einen Rahmen definiert, somit sieht man diese auch später in der Darstellung der MBTextbox nicht mehr. Die Ecken

dieses Steuerelementes werden mit Hilfe eines RoundedRectangle Objekt abgerundet dargestellt. Das so entstandene Punktefeld wird einmal mit der Farbe der TextBox gefüllt und dann bekommt es noch eine dunkle Umrandung wie Abbildung 22 zeigt.

Abbildung 22 - Mühlbauer TextBox

Da die MBTextbox auch ein OPC Item enthält, welches den Text setzen kann muss auch ein Schreiben des Textes in das OPC Item implementiert werden. Hierfür wurde das KeyPress-Event der Standard TextBox benutzt. In diesem wird überprüft ob die Enter-Taste gedrückt wurde. Wenn diese gedrückt wurde und das OPC Item verbunden ist, wird der aktuelle Text in das OPC Item geschrieben und somit an die Maschinensoftware übergeben.

5.4. Die MBLamp

Für dieses Steuerelement mussten die Bilder, aufgrund der Quasitransparenz und das OPC Aktualisierungsverhalten, wie bei dem MBButton angepasst werden.

Dieses Steuerelement kann genutzt werden um einen Sensorenstatus der Maschine grafisch darzustellen. Um verschiedene Farben darzustellen wird, ähnlich wie beim MBButton, eine Enumeration für die Farben definiert. Die Lampe hat zusätzlich eine Eigenschaft LampSize. Damit kann der Entwickler eine vordefinierte Größe für die MBLamp wählen. Zusätzlich wird rechts neben der Lampe ein Text dargestellt. Die Lampe verfügt über einen integrierten Timer, welcher mittels zweier Eigenschaften, Flash und Flashintervall, parametrisiert werden kann. Damit kann man die Lampe ohne OPC Verbindung blinken lassen. Das heißt die Lampe hat zwei verschiedene Stati, an und aus. Für jeden Status und jede Farbe muss ein eigenes Bild gezeichnet werden, wie in Abbildung 23 zu sehen ist.

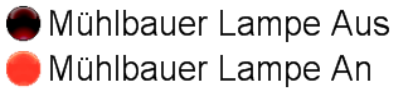

Abbildung 23 - Mühlbauer Lampe

5.5. Die MBGroupBox

Im .Net Framework wurde die MBGroupBox von der Mutterklasse MBControl abgeleitet. Im .Net CF funktionierte das nicht da die MBControl_CE Klasse keine Möglichkeit besitzt anderen Steuerelementen ein Container zu sein. Deshalb wurde hier die MBGroupbox von dem Panel-Element abgeleitet. Dieses Steuerelement hat die Möglichkeit andere Steuerelemente in einem Container zusammenzufassen. Der neu entstandenen Klasse werden zusätzlich noch einige Attribute beigefügt. Das Panel-Element enthält kein Text Attribut deshalb muss dieses extra zu der MBGroupbox hinzugefügt werden, welches hier Headline genannt wurde. Desweiteren musste auch die Schriftart als Eigenschaft (Font) hinzugefügt werden. Auch die Sprachschnittstelle wurde extra hinzugefügt, damit die Sprache dynamisch angepasst werden kann. Eine OPC Verbindung für dieses Steuerelement ist nicht nötig, da es nur dazu dient, Gruppen von Steuerelementen zu erzeugen und visuell darzustellen. Das heißt es musste nur noch die MBRenderer-Methode angepasst werden.

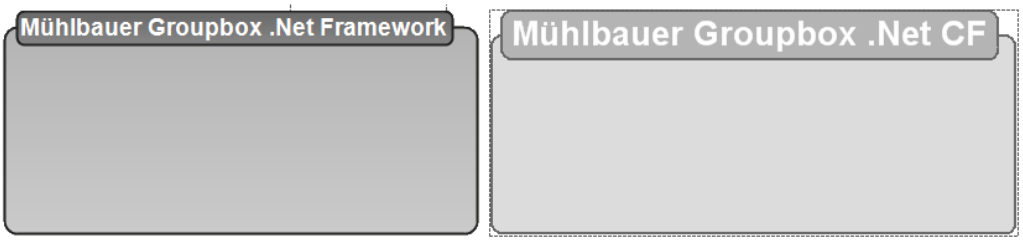

Abbildung 24 - Mühlbauer GroupBox

5.6. Der MBDivider

Damit Steuerelemente auch ohne die MBGroupBox sichtbar gruppiert werden können wurde der MBDivider entwickelt. Um auszuwählen welche Art der Darstellung zu zeichnen ist, wurde im MBRenderer ein neues Attribut angelegt, DividerBorders, auf welches der MBDivider Zugriff hat. Folgende Typen sind möglich:

- None
- Top
- Right
- Left

- Bottom
- Complete
- UnderlineText
- CompleteWithHardEdges

Um die Darstellung richtig zu parametrisieren stehen weitere Eigenschaften zur Verfügung (BorderStyle, BorderColor, BorderWidth). Zusätzlich wurde ein MBLabel zu dem MBDivider hinzugefügt, dessen Eigenschaften wie die Sprachparameter und der Text öffentlich zugänglich gemacht wurden. Eine weitere Eigenschaft zum Anpassen der Position des Labels wurde realisiert. Eine OPC Verbindung für dieses Steuerelement ist nicht notwendig, da es nur zur visuellen Gruppierung von Steuerelementen gedacht ist. In Abbildung 25 werden die möglichen Einstellungen für den BorderStyle dargestellt.

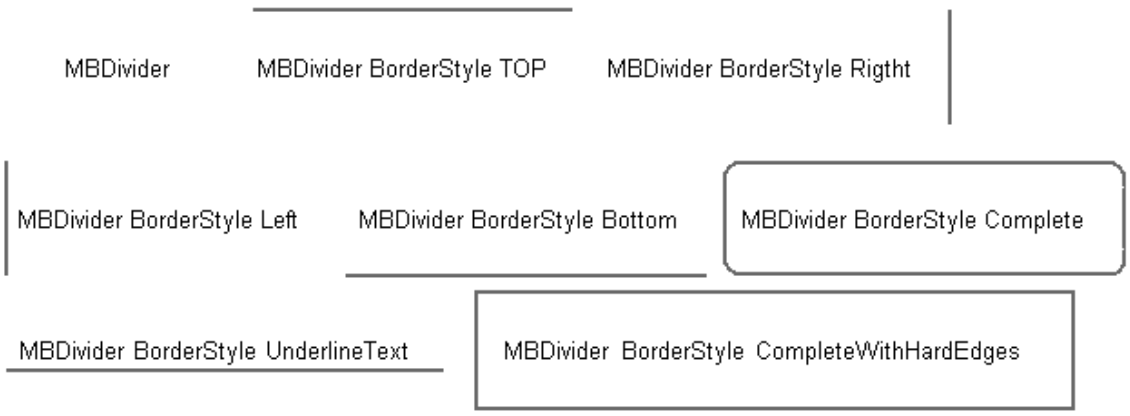

Abbildung 25 - Darstellungsformen des Mühlbauer Divider

5.7. Das MBNumericUpDown

Dieses Steuerelement ist dem Standard NumericUpDown-Element nachempfunden. Es besteht aus einer Textbox und zwei Buttons zum auf und abwärts zählen. Das Prinzip basiert darauf, dass eine Zahl in der Textbox dargestellt wird und diese entweder durch direkte Eingabe gesetzt oder mit Hilfe des Plus- und Minus-Buttons verändert wird. Um die Grenzen für diesen Wert einzustellen werden zwei Attribute, Minimum und Maximum, hinzugefügt. Auch der aktuelle Wert wird in einem Attribut abgespeichert. Um ein so genanntes SpeedUp Gefühl, für den Benutzer, beim auf bzw. abwärts Zählen zu erhalten wurde zusätzlich ein Timer integriert. Ein weiterer

Timer wurde hinzugefügt, um eine Sendeverzögerung einzustellen. Diese Sendeverzögerung ist, wie sich herausgestellt hat, für die OPC Kommunikation sehr von Vorteil. Damit kann sichergestellt werden, dass der Wert erst übernommen wird wenn der Benutzer seine Eingabe beendet hat und nicht permanent auf das OPC Item geschrieben wird. Für den Wert der gezählt werden soll wurde der Typ Double verwendet um einen möglichst großen Zahlenbereich zu erreichen. Damit lassen sich auch Gleitkomma Zahlen darstellen und zählen. Desweiteren erhält dieses Steuerelement folgende Eigenschaften:

- Format, damit kann zwischen dezimaler und hexadezimaler Darstellung umgeschaltet werden
- Increment, gibt die Schrittweite an
- DecimalPlaces, gibt die Anzahl der anzuzeigenden Nachkommastellen an
- Maximum und Minimum
- UseSendDelay, gibt an ob der Wert mit einer Verzögerung übernommen werden soll
- SendDelay_ms, gibt an mit welcher Verzögerung der Wert übernommen werden soll

Falls eine Verzögerung eingestellt ist ändert der Text in der Textbox die Farbe nachdem eine Veränderung des Wertes vorgenommen wird. Erst nach dem der Wert übernommen wurde wird die Farbe wieder zurückgesetzt.

Abbildung 26 - Mühlbauer NumericUpDown

5.8. Die MBCombobox

Die MBCombobox der vorhandenen Steuerelementebibliothek für das .Net Framework besteht aus einer Textbox, einem Button und einem ToolStripDropDown Objekt.

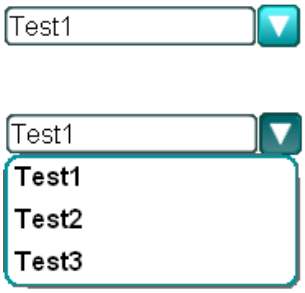

Abbildung 27 - Mühlbauer Combobox .Net Framework

Abbildung 27 zeigt einmal die geschlossene (oben) und die offene (unten) Combobox. Das Feld in dem Test1, Test2 und Test3 zu sehen ist, ist das ToolStripDropDown Objekt. Dieses ToolStripDropDown steht nur im .Net Framework zur Verfügung. Im .Net CF musste es durch ein ContextMenu Objekt ersetzt werden. Von diesem kann die Farbe nicht geändert und die Zeichenmethode kann auch nicht überladen werden. Deshalb konnte die Darstellung des ContextMenu Objektes nicht angepasst werden. Für den Inhalt der MBCombobox steht ein Dictionary, eine Speicherstruktur aus dem .Net CF und .Net Framework, zur Verfügung.

Ein Dictionary ist eine Auflistung von Schlüsseln und Werten [1]. Die Schlüssel und die Werte können Objekte von allen Typen sein. Ein Schlüssel darf nur einmal vorhanden sein. Der Typ, für den Schlüssel und den Wert, wird beim Erzeugen eines Dictionaries angegeben. Um Schlüssel und Werte zu dem Dictionary hinzuzufügen steht die Methode Add zur Verfügung, an diese wird ein Schlüssel und ein Wert übergeben. Um Werte zu löschen wird die Remove Methode aufgerufen, welche nur einen Schlüssel benötigt. Die Adressierung des Dictionaries basiert auf den Schlüsseln das heißt, es können über die Schlüssel die dazugehörigen Werte des Dictionaries ausgelesen oder geändert werden. In Abbildung 28 ist ein kurzes Beispiel für den Umgang mit Dictionaries zu sehen.

```
private Dictionary<String, Object> m_item = new Dictionary<String, Object>();
//Werte zum Dictionary hinzufügen
m_item.Add("Test1", null);
m_item.Add("Test2", new MBButton());
m_item.Add("Test3", 5);
//Werte aus dem Dictionary entfernen
m_item.Remove("Test1");

//einen bestimmten Wert aus dem Dictionary lesen
object o = m_item["Test1"];
MBButton tempbutton = m_item["Test2"];
int fünf = m_item["Test3"];
```

Abbildung 28 - Beispiel für Dictionary

Um ein Dictionary als Eigenschaft (Items) zu einem Steuerelement hinzuzufügen, muss eine Get-Methode erstellt und das gesamte Handling für das Dictionary innerhalb dieser Methode implementiert werden. Eine Set-Methode ist für die Speicherstruktur nicht möglich.

In der Get-Methode wird das ContextMenu geleert und für jeden Schlüssel des Dictionary wird ein neuer ContextMenu-Eintrag erstellt. Zusätzlich wird jedem Eintrag ein Click-Event hinzugefügt, um später den ausgewählten Eintrag zu ermitteln. Wenn der Button gedrückt wird, wird überprüft ob alle Einträge aus dem Dictionary auch in dem ContextMenu stehen. Ist das nicht der Fall, werden die Fehlenden noch hinzugefügt. Danach wird geprüft ob überhaupt Einträge vorhanden sind, wenn ja dann wird das ContextMenu unterhalb der Textbox geöffnet. Der Benutzer kann nun einen Eintrag auswählen. Ist dies erfolgt wird das ContextMenu geschlossen, das gewählte Objekt in der Textbox dargestellt und der Text des Objektes auf ein Attribut der Combobox geschrieben. Zusätzlich wird noch ein Event erzeugt, so dass man auch an anderen Stellen im Code die Information bekommt das ein Eintrag ausgewählt wurde und um welchen es sich handelt. Durch das Dictionary hat man die Möglichkeit jedem Eintrag noch ein bestimmtes Objekt mit zu geben. Die MBCombobox wurde zusätzlich mit der Eigenschaft Editable erweitert wodurch während der Laufzeit ein Text in die Textbox geschrieben werden und mit „Enter“ bestätigt werden kann. Daraufhin wird ein neuer Eintrag in dem Dictionary gespeichert und im Contextmenü erstellt. Abbildung 29 zeigt die neue MBCombobox

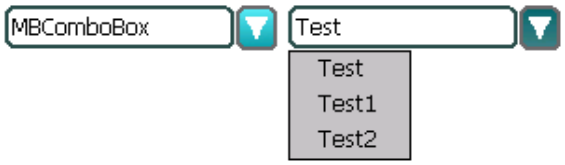

Abbildung 29 - Mühlbauer Combobox .Net CF

5.9. Die MBErrorMessageBox

Um Fehler oder Warnungen aus der Maschinensoftware oder aus dem GUI darzustellen wurde die MBErrorMessageBox entwickelt. Diese besteht aus einem Titel und einem Fehlertext und wurde vom UserControl abgeleitet, da eine OPC – Verbindung nicht notwendig ist. Desweiteren gibt es die Möglichkeit, beim Auftreten mehrerer Fehler oder Warnungen, diese mittels Navigationsbuttons durchzuschalten oder die Meldung zu minimieren. Zusätzlich bekommt der Nutzer noch über eine blinkende Anzeige mitgeteilt ob es sich um einen Fehler oder eine Warnung handelt. Für diese blinkende Anzeige steht der MBErrorMessageBox ein Timer zur Verfügung. Um die Anzeige einzustellen steht die Eigenschaft MessageType zur Verfügung, welche dann je nach Typ die Farbe dieser Anzeige einstellt. Über dem Help Button kann man einen zusätzlichen Hilfe Text anzeigen lassen. Die MBErrorMessageBox ändert dann automatisch die Größe und zeigt eine zusätzliche GroupBox mit dem zugehörigen Hilfetext an. Dadurch dass in dem Timer-Callback immer wieder das Steuerelement neu gezeichnet wird entsteht ein flackern des gesamten Elementes. Um dieses Flackern zu reduzieren bzw. ganz zu entfernen gibt es eine Möglichkeit, welche im Punk 5.10. erläutert wird.

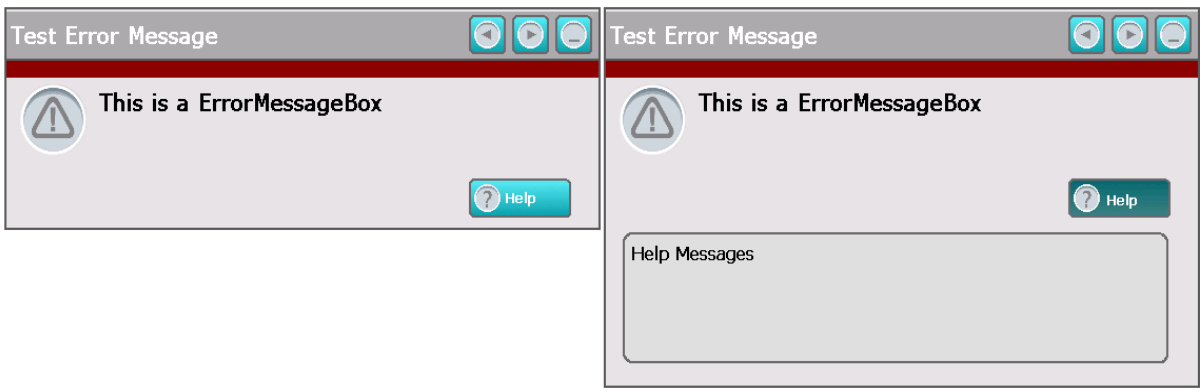

Abbildung 30 - Mühlbauer ErrorMessageBox

5.10. Verbesserung der dynamischen Darstellung

Um das Flackern, bei der dynamischen Darstellung von Steuerelementen zu beiseitigen wird zunächst untersucht wie dies im .Net Framework realisiert ist. Danach wird eine Implementierung für das .Net CF realisiert.

Im .Net Framework, gibt es für Steuerelemente das Attribut DoubleBuffered. Dieses gibt an, ob das Steuerelement seine Oberfläche unter Verwendung eines sekundären Puffers zeichnen soll, um so Flackern zu verringern oder zu vermeiden.

Diese Eigenschaft existiert im .Net CF nicht, sie kann aber simuliert werden. Für diese Simulation darf die Zeichenroutine nicht wie üblich direkt auf das Steuerelement zugreifen sondern zeichnet zunächst alles auf ein Bitmap Objekt. Von diesem Bitmap Objekt wird dann ein Graphicsobjekt erzeugt. Mit diesem Graphicsobjekt kann man dann alle Operationen durchführen. Das heißt alle Zeichenoperationen werden auf dem Graphicsobjekt ausgeführt also auf dem virtuellen Bitmap Objekt. Wenn alle Zeichenoperationen beendet sind muss man dieses Bitmap Objekt, nur einmal zeichnen, wie in Abbildung 31 veranschaulicht ist.

```
Bitmap m_bmpOffScreen;
protected override void OnPaint(PaintEventArgs e)
{
    base.OnPaint(e);
    Graphics gxoff;
    if (m_bmpOffScreen == null || m_bmpOffScreen.Size != ClientSize)
        m_bmpOffScreen = new Bitmap(ClientSize.Width,
            ClientSize.Height);
    gxoff = Graphics.FromImage(m_bmpOffScreen);
    //...perform your Drawingcode on the gxoff object
    //at the end draw the bitmap on the control
    e.Graphics.DrawImage(m_bmpOffScreen, 0, 0);
}
```

Abbildung 31 - Zeichnen auf Bitmap

Zusätzlich muss man noch die OnPaintBackground Methode, über welche jedes Control und UserControl im .Net CF verfügt, überladen und innerhalb dieser Methode keinen Code ausführen, wie Abbildung 32 verdeutlicht.

```
protected override void OnPaintBackground( PaintEventArgs e)
{ }
```

Abbildung 32 - Überladene OnPaintBackground Methode

5.11. Die MBProgressBar

Auch dieses Steuerelement wurde vom UserControl abgeleitet. Mit Hilfe der MBProgressBar kann ein Statusverlauf dargestellt werden. Über eine Eigenschaft kann man die Hintergrundfarbe der MBProgressBar einstellen und man hat die Möglichkeit den Style zu ändern.

- Continous, der aktuelle Stand der MBProgressBar muss durch den Entwickler gesetzt werden.
- Marquee, ein grafischer Balken, welcher über Attribute eingestellt werden kann, wird automatisch vom Anfang der MBProgressBar bis zum Ende durchgeschoben und beginnt wieder von vorne.
- Knight Rider, dieser Style definiert einen Balken der vom Anfang bis zum Ende der MBProgressBar läuft und wieder zurück.

Es wurde noch ein Label zu der MBProgressBar hinzugefügt, welches den aktuellen Stand der MBProgressBar in Prozent angibt. Diese Anzeige kann nur mit den Continous-Style dargestellt werden, ist aber auch ganz abschaltbar.

Der Marquee und der Knight Rider Style werden über eine Timer gesteuert, welcher dann auch das gesamte Element neu zeichnen lässt, wodurch wie bei der MBErrorMessageBox ein Flackern entsteht. Dieses Flackern wird wie unter 5.10. beschrieben reduziert bzw. entfernt. In Abbildung 33 ist die MBProgressBar mit dem ContinousStyle zu sehen.

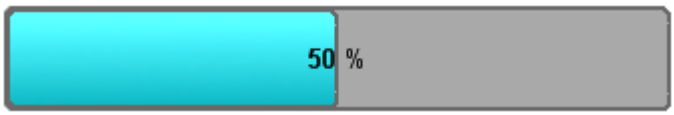

Abbildung 33 - Mühlbauer Progress Bar

5.12. Der MBLampButton

In der Steuerelementebibliothek für das .Net Framework ist dieses Element aus der MBLamp und dem MBButton zusammengesetzt. Aufgrund der fehlenden Transparenz im .Net CF und Windows CE ist dies nicht möglich da man als Hintergrund für die MBLamp keinen Gradienten wie auf dem MBButton nutzen kann.

Deshalb muss im .Net CF dieses Steuerelement umgestaltet werden. Das heißt zu dem Normalen Button wird einfach das Verhalten einer MBLamp implementiert.

Der MBLampButton ist abgeleitet vom MBButton und enthält zusätzliche Eigenschaften um eine Lampe zu realisieren.

- Zwei Lampenbilder (für An und Aus) und eine boolesche Variable für diesen Status
- Die Lampenfarbe
- Die Lampengröße
- Ein OPC Item für den Lampenstaus
- Die Ausrichtung der Lampe auf dem Button
- Eine Variable um den Status der Lampe zu invertieren

Um in diesem Steuerelement noch das OPC Item für die Lampe zu implementieren wurde die OverridableAddToOPCGroup Methode aus der MBControl_CE-Mutterklasse überladen. Desweiteren wurde dem OPC Item eine Aktualisierungsmethode hinzugefügt, in der man wieder auf die Synchronisation des OPC Threads und des MBLampButton Threads achten muss, um Statusänderungen sichtbar zu machen. Auch die OnPaint-Methode musste überladen werden, um die Lampe zusätzlich auf dem Button darzustellen. Um auch das Zeichnen des Buttons zu gewährleisten muss beim Eintritt in diese Methode die Zeichen Methode der Elternklasse gerufen werden, wie in Abbildung 34 dargestellt ist.

```
protected override void OnPaint(PaintEventArgs e)
{
    base.OnPaint(e); //Aufruf der Paint-Methode der Elternklasse
    bool bStateToShow;
    //Überprüfung ob die Lampe invertiert werden soll
    if (m_bLampInverted)
        bStateToShow = !m_LampOn;
    else
        bStateToShow = m_LampOn;
    //Auswahl ob die Lampe an oder aus ist
    Image drawingImg;
    if (bStateToShow)
        drawingImg = m_LampOnPicture;
    else
        drawingImg = m_LampOffPicture;
    MBRenderer.RenderLampButtonLamp(e.Graphics, this.ClientRectangle,
      drawingImg, this.LampAlignment, m_LampRect, 0);
}
```

Abbildung 34 - MBLampButton - OnPaint Methode

In der Methode RenderLampButtonLamp aus der MBRenderer-Klasse, wird anhand der Eingangsparameter das Lampenbild gezeichnet. Abbildung 35 zeigt des kompletten MBLampButton.

Abbildung 35 - Mühlbauer LampButton

5.13. Das MBPopupWindow

Hardwarebedingt kann nur eine maximale Auflösung von 640 x 480 Pixel dargestellt werden. Dadurch ist nur eine begrenzte Anzahl von Steuerelementen auf einem Fenster möglich. Um dennoch eine größere Menge an Steuerelementen darzustellen, wurde das MBPopupWindow entwickelt. Es ist vom UserControl abgeleitet, um das Hinzufügen von Steuerelementen zu zulassen und damit es später beim Eintreten von Events angezeigt werden kann. Das MBPopupWindow beinhaltet noch Eigenschaften wie einen Titeltext, das PopupOpened Event, das PopupClosed Event und einen Close Button.

Wenn man ein MBPopupWindow benutzen möchte erstellt man ein neues UserControl und leitet dieses dann von MBPopupWindow ab. Zum Anzeigen eines MBPopupWindow wird die Methode ShowPopupOn gerufen. Diese Methode benötigt ein Steuerelementobjekt als Übergabeparameter auf dem das MBPopupWindow dargestellt werden soll. Das MBPopupWindow deaktiviert beim anzeigen automatisch alle Elemente, die sich auf dem Elternsteuerelement befinden, damit nur noch der Zugriff auf das MBPopupWindow erlaubt ist. Beim Schließen dieses Elementes werden wieder alle Elemente des Elternsteuerelementes aktiviert. Abbildung 36 zeigt das MBPopupWindow.

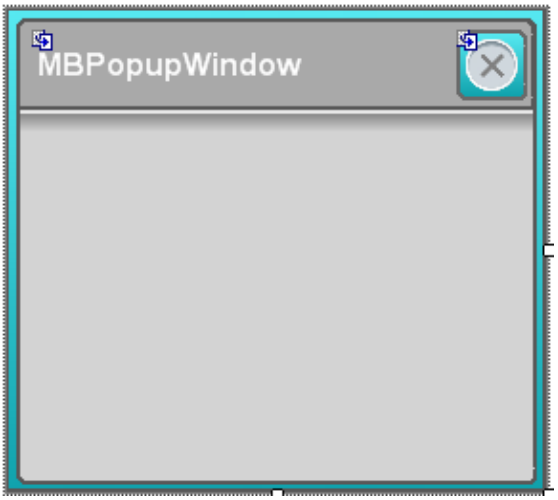

Abbildung 36 - Mühlbauer PopupWindow

5.14. Der MBCheckBoxButton

Der MBCheckBoxButton ist dem Standard Checkbox Element nachempfunden. Er besteht aus einem Rechteck, in dem je nach Checkstatus ein Hacken oder nichts dargestellt wird und einem Text welcher vor oder hinter dem Rechteck stehen kann. Zusätzlich hat er noch die Attribute Checked, welches den aktuellen Status in Form einer booleschen Variable zurückliefert und UseAsRadioButton, um alle MBCheckBoxButtons die sich auf einem Element befinden als RadioButton nutzen zu können. Um die UseAsRadioButton Eigenschaft zu realisieren muss in der Set-Methode für das Checked Attribut die Variable UseAsRadioButton überprüft werden. Dabei wird überprüft ob sich auf dem Eltern-Element weitere MBCheckBoxButtons befinden, bei welchen das Attribut UseAsRadioButton gesetzt ist. In diesem Fall wird das Checked Attribut der anderen MBCheckBoxButton so angepasst das nur ein MBCheckBoxButton aktiviert sein kann. Abbildung 37 zeigt den entsprechenden Code dafür und Abbildung 38 die Darstellung des MBCheckBoxButtons.

```
public bool Checked
{
    get { return m_Checked; }
    set
    {
        //Set the actual value to the attribute
        m_Checked = value;
        //check if the control is checked and
        //it should be used as Radiobutton
        if (m_Checked && m_UseAsRadioButton)
        {
            //check if a parent control exits
            if (this.Parent != null)
            {
                //a loop over all controls on the parent
                foreach (Control c in this.Parent.Controls)
                {
                    //search for MBCheckBoxButtons
                    if (c.GetType() == typeof(MBCheckBoxButton))
                    {
                        MBCheckBoxButton temp = c as MBCheckBoxButton;
                        //check if the UseAsRadioButton is true
                        if (temp != this && temp.UseAsRadioButton)
                        {
                            temp.Checked = false; ;
                            if (temp.m_OPCItem.InGroup)
                                temp.m_OPCItem.WriteValue(false);
                        }
                        if (temp == this)
                            temp.CheckChanged(this, EventArgs.Empty);
                    }
                }
            }
        }
        if (!m_UseAsRadioButton)
        {
            //throw CheckChanged event
            CheckChanged(this, EventArgs.Empty);
        }
        CalculateSizes();
        this.Invalidate();
    }
}
```

Abbildung 37 - MBCheckBoxButton - UseAsRadioButton Code

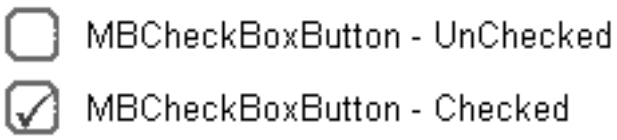

Abbildung 38 - MBCheckBoxButton

5.15. Das MBFlowLayoutPanel

Im .Net Framework existiert das FlowLayoutPanel als Standard Steuerelement, welches automatisch alle Steuerelemente die sich auf ihm befinden nach bestimmten

Vorgaben anordnet. Im .Net CF existiert ein solches Element nicht und wurde deshalb neu entwickelt.

Abgeleitet ist das MBFlowLayoutPanel vom Standard Panel aus dem .Net CF um somit, wie die MBGroupbox, als Container für andere Steuerelemente zu dienen. Es wurden drei Attribute zu diesem Element hinzugefügt, ein LayoutStyle in Form einer Enumeration, eine Integer Variable ControlDistance und eine boolesche Variable AutoLayout. Der LayoutStyle gibt an wie das MBFlowLayoutPanel seine Elemente anordnen soll.

- None, die Elemente werden nicht ausgerichtet
- LineFlowLayout, die Elemente werden nur auf einer Zeile ausgerichtet, die Größe des MBFlowLayoutPanel bleibt unverändert
- LineFlowLayoutAndResize, die Elemente werden auf einer Zeile ausgerichtet und das MBFlowLayoutPanel passt seine Breite automatisch an die Anzahl und die Größe seiner Elemente an
- CompleteFlowLayout, die Elemente werden innerhalb des gesamten MBFlowLayoutPanels ausgerichtet

Die ControlDistance gibt den Abstand zwischen den Elementen an und das AutoLayout ob die Steuerelemente automatisch angeordnet werden sollen.

Für das Ausrichten der Steuerelemente wird die Methode Layout implementiert. Diese ist auch außerhalb des MBFlowLayoutPanels aufrufbar um die Steuerelemente erneut anzuordnen. In dieser Methode wird die LayoutStyle Eigenschaft überprüft. Je nachdem welcher LayoutStyle eingestellt wurde, werden die Steuerelemente auf dem MBFlowLayoutPanel angeordnet.

Abbildung 39 zeigt einen Codeausschnitt für den LayoutStyle LineFlowLayoutAndResize.

```
Int nextTop = ControlDistance, nextLeft = ControlDistance;
foreach (Control myControl in this.Controls)
{
    myControl.Top = nextTop;
    myControl.Left = nextLeft;
    nextLeft += myControl.Width + ControlDistance;
    if (nextLeft > this.Width)
        this.Width = nextLeft;
}
```

Abbildung 39 - Codeausschnitt für LineLayoutAndResize

Die Variablen nextTop und nextLeft werden mit der ControlDistance initialisiert und alle Steuerelemente die sich auf dem MBFlowLayoutPanel werden neu positioniert. Jedes Element wird auf die Position von nextTop und nextLeft gesetzt. Dann wird zu der nextLeft Variable die aktuelle Elementenbreite und die ControlDistance addiert. Falls nextLeft jetzt größer ist als die Breite des MBFlowLayoutPanels wird diese auf nextLeft gesetzt um somit die Breite den Elementen anzupassen. Abbildung 40 zeigt drei verschiedene LayoutStyles, links None, rechts CompleteFlowLayout und unten LineFlowLayoutAndResize.

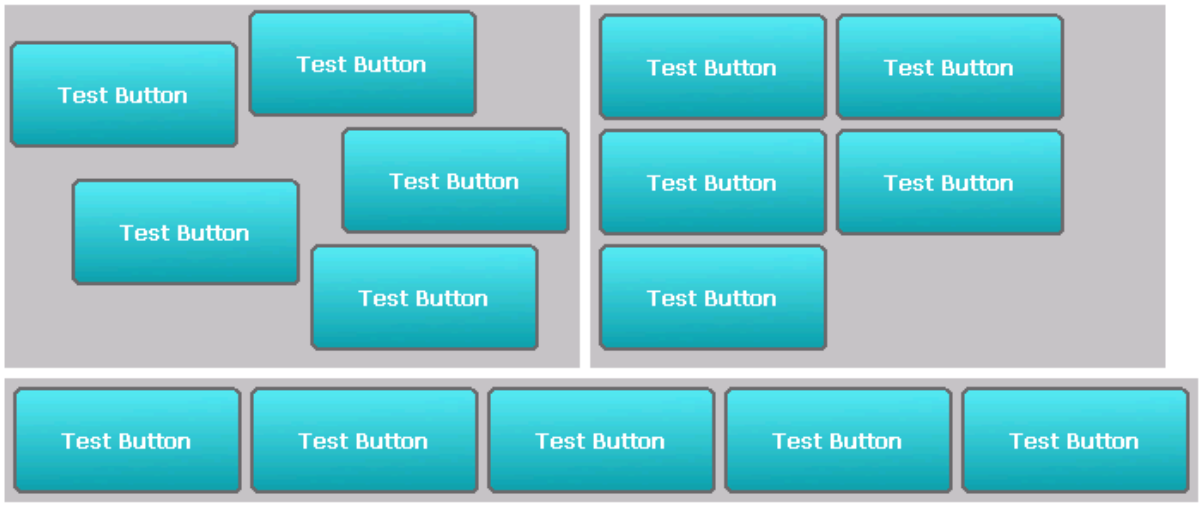

Abbildung 40 - Mühlbauer FlowLayoutPanel

6. Kombinierte Steuerelemente

6.1. Die zweite Basisklasse MBM2kHardwareControl

Diese Klasse wird als Mutterklasse für die kombinierten Steuerelemente genutzt, um Hardwarebaugruppen über OPC zu steuern. so wird auch hier wieder eine Grundfunktionalität implementiert. Diese Klasse ist abgeleitet vom UserControl um später auch andere Steuerelemente aufnehmen zu können. Sie verfügt über zwei Attribute für die OPC Steuerung, einen OPC Pfad und eine boolesche Variable UseDefaultOPCNames. Beide Attribute werden zu Eigenschaften dieser Klasse gemacht und zusätzlich wird noch eine Zeichenkette hinzugefügt, OPCTagWithDot, der den aktuellen OPC Pfad mit einem Punkt am Ende zurück liefert. Auch hier existiert wie in der MBControl_CE Klasse eine Methode AddToOPCGroup. Allerdings

ist diese hier virtuell und muss in den abgeleiteten Steuerelementen überladen werden, da die kombinierten Steuerelemente meistens aus mehreren Elementen mit OPC Verbindung bestehen. Zusätzlich wurde auch hier die Sprachschnittstelle mit integriert, so dass man die Eigenschaft Headline dynamisch mit der aktuellen Sprache laden kann. Diese Klasse wird als MBGroupBox gezeichnet, um somit einen visuellen Rahmen für die kombinierten Elemente zu erhalten.

6.2. Der MBCylinder

Dieses Steuerelement ist zur Steuerung eines Pneumatiksteuerzylinders gedacht. Es besteht aus zwei MBLampButtons und einem Label um die Zeit der Zylinderbewegung anzuzeigen. Da es verschiedene Zylindertypen gibt soll die Darstellung des MBCylinder auch veränderbar sein. Dafür gibt es eine Enumeration mit verschiedenen Zylindertypen.

- Horizontal
- HorizontalInverted
- Vertical
- VerticalInverted
- Turning

Diese Enumeration ist an verschiedene Bilder für die Buttons gekoppelt um somit die Darstellung anzupassen, was in Abbildung 41 zu sehen ist.

Abbildung 41 - Mühlbauer Zylinder

Ein Zylinder besteht in der Maschinensoftware aus mehreren OPC Einträgen, welche man in diesem Steuerelement einstellen kann. Das heißt für jeden Pfad gibt es eine String Variable um diesen einzustellen. Die Lampen zeigen den Status eines Sensors an, das heißt in welcher Position der Zylinder steht. Mit dem ButtonClick-Event wird die Position geändert.

6.3. Der MBDCMotor

Der MBDCMotor ist zur Steuerung eines DC-Motors gedacht. Diese Art von Motoren können vorwärts oder rückwärts fahren und stoppen. Zusätzlich besteht die Möglichkeit den Motor auf einen Sensor anzuhalten. Um dies in dem Steuerelement zu realisieren sind drei MBButtons und ein MBCheckBoxButton nötig. Um auch dieses Steuerelement so flexibel wie möglich zu halten, kann man für jede Aktion, also fahre vorwärts, fahre rückwärts, stoppe und stoppe auf einen Sensor, jeden OPC Pfad selbst einstellen. Zusätzlich kann man die Richtungsbilder auf den Buttons ändern oder einstellen, so dass sie getauscht werden. Außerdem kann man dieses Steuerelement so konfigurieren das nur eine Richtung gefahren kann, also nur ein Richtungsbutton und der Stoppbutton werden dargestellt.

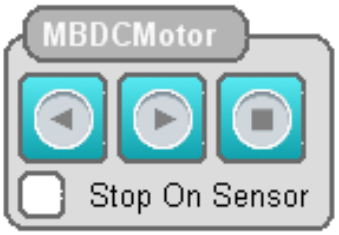

Abbildung 42 - Mühlbauer DCMotor

6.4. Der MBStepperSimple

Da in der Maschinensoftware verschiedene Implementierungen für Schrittmotoren vorhanden sind, gibt es auch zwei verschiedene kombinierte Steuerelemente dafür. Der MBStepperSimple besteht aus einer MBLamp, zwei MBNumericUpDowns und noch drei MBButtons zum vorwärts und rückwärts fahren und um den Motor zu stoppen.

Die MBLamp zeigt einen Sensorwert an welcher angibt ob der Schrittmotor gestoppt ist oder läuft. Mit den MBNumericUpDowns stellt man die Geschwindigkeit des Motors und den Weg ein den der Steppermotor fahren soll. Über die Buttons wird der Motor gestartet und gestoppt. Wie auch in den anderen kombinierten Steuerelementen kann man hier die OPC Pfade für jeden Befehl einstellen und einstellen ob die Darstellung der Bilder auf den vorwärts und rückwärts Button invertiert sein soll. Abbildung 42 zeigt das MBStepperSimple Element.

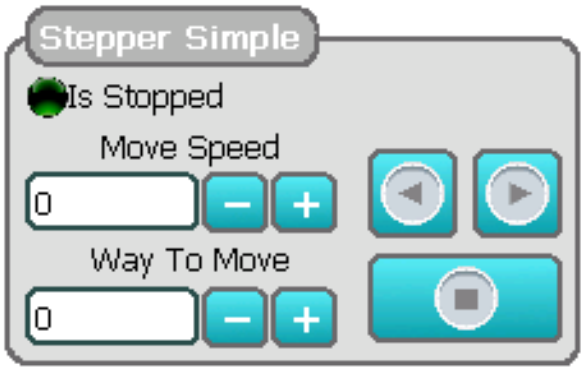

Abbildung 43 - Mühlbauer StepperSimple

6.5. Der MBStepper

Der MBStepper ist dem MBStepperSimple ähnlich, der Unterschied zwischen beiden ist das der MBStepper mehr Parameter zum einstellen hat und zusätzlich noch mehr Sensoren anzeigt. Zusätzlich gibt es die Möglichkeit zu einer bestimmten Position zufahren. Auch in diesem Steuerelement kann mal alle OPC Pfade über Eigenschaften konfigurieren.

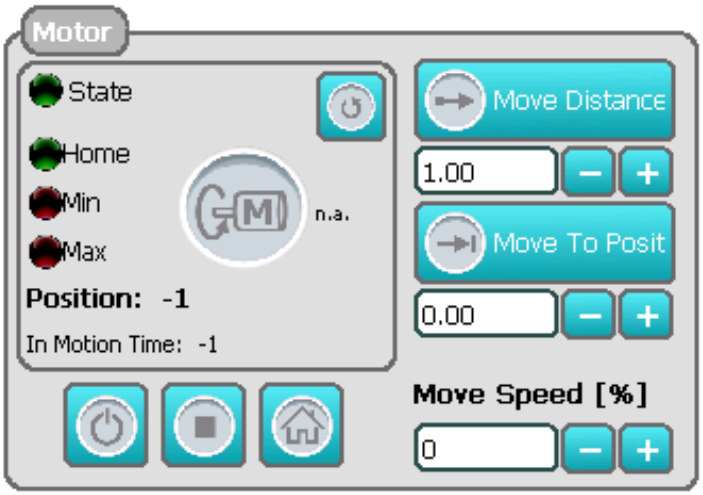

Abbildung 44 - Mühlbauer Stepper

7. Tests

Zur Gewährleistung der Funktionsfähigkeit und der Kontrolle der Darstellung der Steuerelemente ist eine Testphase notwendig.

7.1. Testarten

Es lassen sich vier Testarten für die Steuerelemente definieren.

- Grafiktest
- Funktionstest
- Systemtest

Grafiktest

Bei diesem Test wird nur das Aussehen der Steuerelement untersucht, dass heißt es wird auch nur der Teil mit der grafischen Ausgabe in das Steuerelement implementiert. Das Steuerelement bekommt keinerlei Funktionalität. Hierbei wird auf Abstände, Grafikfehler oder Überlappungen der Grafiken geachtet.

Funktionstest

Um die Funktionalität des entwickelten Steuerelementes zu überprüfen wird der Funktionstest durchgeführt. Hierbei besteht die Notwendigkeit die Parameter und die Randbedingungen zu prüfen. Dafür stehen zwei Testtechnoliegen zur Verfügung, der „Black Box Test“ und der „White Box Test“.

Bei dem „Black Box Test“ werden die erwarteten Resultate der Methoden auf Funktionalität überprüft. Es wird untersucht wie die Methoden auf gültige und ungültige Daten reagiert. Es kann dabei keine Aussage über die Qualität der Programmierung getroffen werden. Bei diesem Test wird jede einzelne Methode eines Steuerelementes, mit bestimmten Eingangsparametern, durchlaufen. Daraufhin wird das Resultat dieser Methode beurteilt.

Beim „White Box Test“ hingegen überprüft man die Struktur der Software, auf „Sauberkeit“ der Programmierung und häufig auftretende Programmierfehler. Die Überprüfung erfolgt teilweise automatisch und ist in Ansätzen in Compilern enthalten.

Die Testdurchführung eines Black Box oder White Box Tests beginnt mit einem Testcase. Ein Testcase ist die Definition aller Schritte, die notwendig sind um für ein Merkmal die Testergebnisse zu ermitteln. Ein Merkmal ist die Ausprägung, die das Endsystem besitzen soll. Die Durchführung des Tests erfolgt in einer definierten Reihenfolge. Dazu werden alle Abhängigkeiten nacheinander aufgelöst und abgearbeitet.

Schlägt ein Test im abzuarbeitenden Testcase fehl, sind die aufgetretenen Fehler zu beseitigen und alle bis dahin durchgeführten Tests zu wiederholen.

Systemtest

Zur Überprüfung des gesamten Steuerelementes wird der Systemtest genutzt. Dabei wird die Gesamtfunktionalität überprüft. Das heißt es werden die Methode mit der grafischen Ausgabe zusammen getestet.

7.2. Durchführung

In der Entwicklungsumgebung MS Visual Studio 2005 wird ein GUI Projekt erstellt. Ein GUI für das .Net Framework und eine für das .Net CF. In diese GUIs werden die einzelnen Steuerelemente integriert. Die so entstandene Anwendung wird einmal unter Windows XP und einmal auf dem entsprechenden Zielsystem ausgeführt. Während der Ausführung dieser GUIs werden die Testarten durchgeführt.

7.2.1. Der Grafiktest

Als Beispiel wird der Grafische Test anhand des MBDividers (siehe 5.6) erläutert. Die Aufgabe des MBDivider besteht darin mittels grafischer Mittel Steuerelemente zu gruppieren oder von einander grafisch zu trennen. Für den MBDivider sind verschiedene Darstellungsarten vorhanden welche unter 5.6 erläutert werden. Nach der Implementierung einer Darstellungsart wird das Steuerelement in die GUI integriert und auf dem entsprechenden Zielsystem getestet. Ist die Darstellung korrekt wird die nächste Methode implementiert und getestet, bis die Darstellung des neuen Steuerelementes der des vorhandenen Elementes entspricht.

Bei anderen Steuerelementen wird immer erst ein Teil der Darstellung implementiert und getestet. Um zu verhindern dass die Zeichenroutine zu groß wird und der Überblick verloren geht.

7.2.2. Der Funktionstest

Ein Testcase wird für jedes Element mit einer OPC Verbindung definiert. Wenn in der Aktualisierungsmethode eine grafische Änderung vollzogen werden muss, muss diese auch sichtbar sein (Threadsynchronisation).

Der Steuerelemente-Eigenschaften Testcase ist, dass sämtliche Attribut welche als Eigenschaft des Steuerelementes definiert werden und über eine erweiterte Set- bzw. Get-Methode verfügen, auf Funktionalität geprüft werden (siehe MBCombobox 5.8, die Get-Methode für das Dictionary).

Beispiel für einen Testcase

Testcase Name:

- Testcase für die MBCombobox Get-Methode

Eingangsparameter:

- Dem Item-Dictionary der MBCombobox wird ein neuer Eintrag, mit einem String Schlüssel und einer Integerzahl als Wert, hinzugefügt

Erwartete Funktion:

- Ein neuer ContextMenueintrag soll erstellt werden
- Der Schlüsselwert soll der Anzuzeigende Name für den Eintrag sein

Resultat:

- Beim ersten Hinzufügen trat das erwartete Ergebnis ein
- Beim zweiten Hinzufügen trat als Fehler auf, dass der Eintrag mit diesem Schlüssel schon vorhanden ist

Lösung:

- Überprüfung ob sich der Schlüssel schon im dem Dictionary befindet, wenn ja abbrechen wenn nein hinzufügen

7.2.3. Der Systemtest

Bei diesem Test wird, während der Laufzeit, das gesamte Steuerelement getestet. Das heißt, die OPC Verbindung, die grafische Darstellung und die Funktionalitäten der Methoden werden getestet. Weiterhin wird geprüft ob das Element richtig auf verschiedene Benutzeraktionen richtig reagiert.

Als Beispiel dafür wird dieser Test kurz anhand des MBNumericUpDowns (siehe 5.7.) erläutert. Hier werden während der Laufzeit die Funktionalitäten der beiden Timer getestet, ob der eingestellte Wert auch in das OPC Item geschrieben wird und ob er auch von dort gelesen werden kann. Es wird auch getestet ob sich die Positionen der MBButtons und der MBTextbox bei Größenänderungen des MBNumericUpDows richtig gesetzt werden und ob der Plus- und Minusbutton korrekt funktioniert.

7.3. Ergebnisse

Jedes Steuerelement wurde auf diese Weise getestet. Bei jedem Steuerelement, welches über eine OPC Verbindung verfügt, musste mindestens die Synchronisation zwischen dem OPC Thread und dem Steuerelement Thread hinzugefügt werden. Desweiteren musste für alle Steuerelemente die Zeichenroutine angepasst werden. Bei Funktionen die erweiterte String-Operationen beinhalteten mussten diese umgeschrieben werden, da im .Net CF die String-Methoden stark reduziert wurden.

Für bestimmte Steuerelemente wie dem MBButton und der MBLamp musste die Quasitransparenz für die Bilder hinzugefügt werden. Andere Elemente mussten komplett neu entwickelt werden, da das ursprüngliche Steuerelement zu viele Methoden und Objekte des .Net Frameworks beinhaltete und somit in seiner Form nicht portierbar war (MBGroupbox, MBCombobox). Es wurden auch Performance Tests durchgeführt, um die schnellste Methode für die Darstellung von grafischer Transparenz zu finden. In der nachfolgenden Tabelle 2 wird gezeigt welche Steuerelemente wie verändert wurden.

Steuerelement	Art der Änderung / Anpassung
MBButton	• Threadsynchronisation • Anpassung der Stati • Quasitransparenten Zeichnen • RoundedRectangle Darstellung
MBLabel	• Threadsynchronisation • DecimalPlaces und RoundingMethod
MBTextbox	• Threadsynchronisation • RoundedRectangle Darstellung
MBLamp	• Threadsychonistion • Quasitransparentes Zeichnen
MBGroupBox	• Neuentwicklung • Ableitung vom Panel – Objekt • Headline Eigenschaft
MBDivider	• Anpassung der Zeichenmethoden
MBNumericUpDown	• Threadsynchronisation • SpeedUp Timer • SendTimer
MBCombobox	• Neuentwicklung • Umstellung auf ContextMenü
MBErrorMessageBox	• Anpassung der Zeichenmethoden • Verminderung von Flackern • Automatisches Vergrößern
MBProgressBar	• Anpassung der Zeichenmethoden
MBLampButton	• Neuentwicklung • Keine Kombination von MBLamp und MBButton • Hinzufügen von zwei Bildern für die Lampe • Threadsynchronisation
MBPopupWindow	• Abgeleitet von UserControl • Hinzufügen von Eigenschaften

	• Neuentwicklung der Zeichenroutine
MBCheckBoxButton	• Threadsynchronisation • Anpassung der Zeichenroutinen
MBFlowLayoutPanel	• Neuentwicklung • Berechnung der Positionen der Elemente
MBCylinder	• Threadsynchronisation
MBDCMotor	• Threadsynchronisation
MBStepperSimple	• Threadsynchronisation
MBStepper	• Threadsynchronisation • Anpassung der Zeichenroutine

Tabelle 2 - Änderungen der Steuerelemente

8. Zusammenfassung und Ausblick

Während dieser Arbeit wurde, gemäß der Aufgabenstellung, eine Steuerelementebibliothek für das .Net CF und Windows CE entwickelt.

Das vorhandene System (die MBControl.dll) wurde in seine Bestandteile untergliedert und analysiert.

Einige Elemente konnten aus der vorhandenen Bibliothek für das .Net Framework unverändert bzw. mit minimalen Anpassungen verwendet werden. Die meisten Steuerelemente mussten neuentwickelt werden, da die dazugehörigen originalen Elemente auf zu viele Methoden oder Attribute des .Net Frameworks zugriffen. Bei der Neuentwicklung wurden weitere Probleme entdeckt, wie zum Beispiel das Fehlen der graphischen Transparenz unter Windows CE und .Net CF. Diese wurde durch das übereinander Zeichnen von zwei Bildern gelöst. Zusätzlich musste ein Weg gefunden werden um Gradienten unter Windows CE darzustellen.

Größtes Augenmerk lag darauf weitestgehende Übereinstimmung im Aussehen und Verhalten der neuen Steuerelemente zu erhalten.

Die Funktionalität und das Aussehen der Steuerelemente wurden bisher nur mit kleinen GUI-Tools getestet.

Künftig wird ein komplett neues GUI, in Anlehnung an das Maschinen-GUI für Windows XP, basierend auf Windows CE und dem .Net CF entwickelt, welche die komplette neue Steuerelementebibliothek nutzen wird.

Quellenverzeichnis

[1] Microsoft. Microsoft Developer Network. MSDN. [Online] Microsoft, 2009
http://www.msdn.com

[2] Andreas Kühnel. Visual C# 2008. Galileo Computing, 4. Auflage
ISBN: 978-3-8362-1172-7

[3] Einführung in die OPC – Technik.
http://www.systech-gmbh.de/produkte/pdf/opc.pdf

Anhang

CDROM Verzeichnisstruktur

Hauptverzeichnis: BA_DanielBaum.pdf – Digitale Bachelorarbeit

Thesen

1. Für die Darstellung von Gradienten existiert im Microsoft .Net Compact Framework keine Funktionalität. Diese muss selbst entwickelt und implementiert werden.

2. In Windows CE ist es nicht möglich Bilder mit Transparenz darzustellen. Um dies dennoch umzusetzen ist es notwendig eine Quasitransparenz zu erschaffen.

3. Um Änderungen der Maschinenprozessvariablen in Steuerelementen sichtbar zu machen muss der OPC-Thread mit dem Thread des jeweiligen Steuerelementes synchronisiert werden.

4. Zur Reduzierung von Flackern bei der Darstellung der Steuerelemente gibt es keine DoubleBuffered Eigenschaft wie im .Net Framework. Diese muss hier neuentwickelt werden.