


## Inhalt

Inhalt .....	1
1 Inhalt .....	2
2 Einführung .....	4
2.1 Einführendes Beispiel .....	5
3 Endlicher Automat .....	8
3.1 Arten .....	11
3.1.1 Moore-Automat .....	12
3.1.2 Mealy-Automat .....	13
3.2 Automaten in der Praxis .....	14
3.2.1 Darstellung in der UML .....	15
3.2.2 Design .....	18
4 Programmierung endlicher Automaten .....	19
4.1 Der Automat .....	20
4.2 Realisierung .....	22
4.2.1 switch-case .....	23
4.2.2 State Pattern .....	25
4.2.3 Tabelle .....	28
4.2.4 Eingebettete FSM .....	49
4.3 Ausblick .....	53
5 Fazit .....	55
6 Download-Link .....	56

# 1 Inhalt

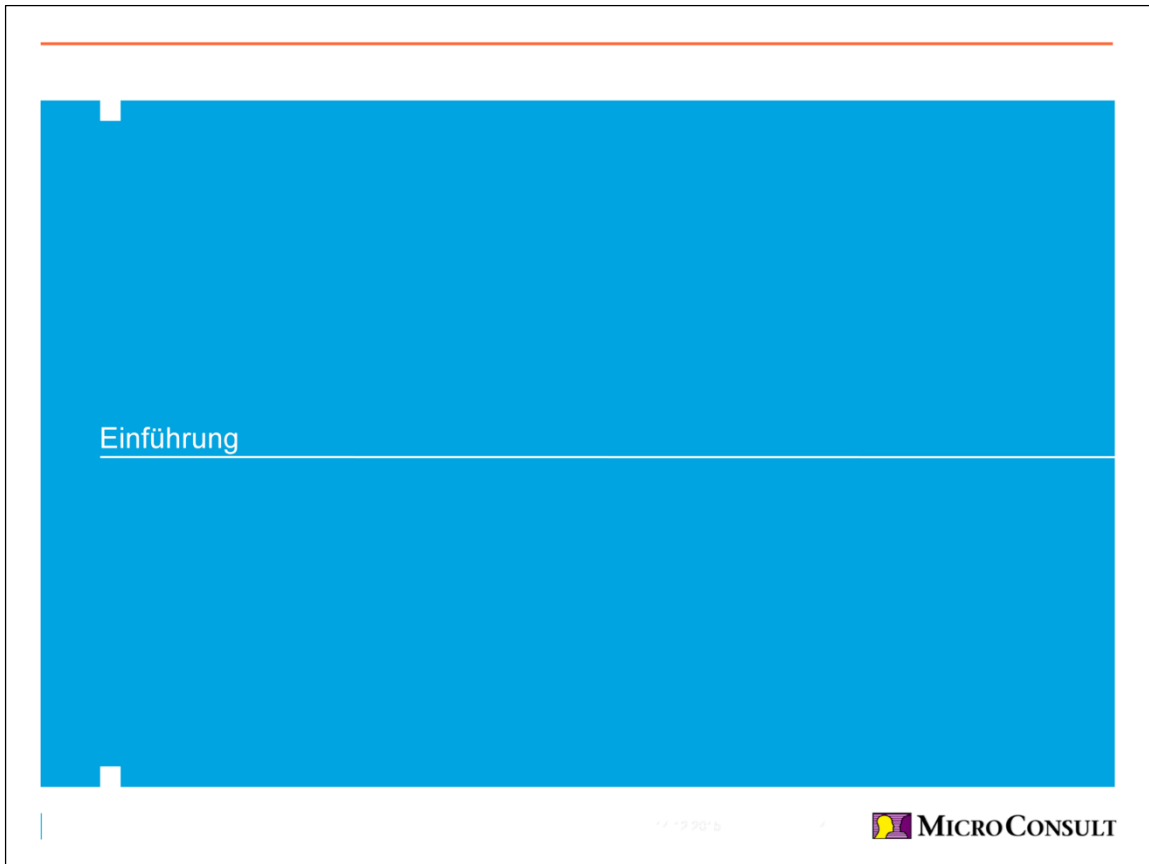
		Inhalt
	Warum werden Automaten eingesetzt?	
	Welche Arten von Automaten gibt es?	
	Wie werden Automaten dargestellt?	
	Was ist beim Design von Automaten zu beachten?	
	Wie können Automaten in C++ implementiert werden?	

© MicroConsult - Microelectronics Consulting & Training GmbH 14.12.2015 F 2 

Im Embedded-Bereich werden viele Applikationen programmiert, die zustandsbehaftete Systeme steuern.

Modellierungstools mit Codegenerierung sind oft zu teuer oder im aktuellen Projekt nicht einsetzbar.

## 2 Einführung



## 2.1 Einführendes Beispiel


Einführung – Einführendes Beispiel

Im Code finden Sie folgende Funktion:

```
int g_nRot= 0;
int g_nGelb= 0;
int g_nGruen= 0;

void process(char c)
{
    if(g_nRot && !g_nGelb)
    {
        g_nGelb= 1;
        GelbEin();
    }
    else if(g_nRot && g_nGelb)
    {
        g_nRot= 0;
        g_nGelb= 0;
        g_nGruen= 1;
        RotAus();
        GelbAus();
        GruenEin();
    }
}
```

```
else if(g_nGruen)
{
    g_nGruen= 0;
    g_nGelb= 1;
    GruenAus();
    GelbEin();
}
else if(g_nGelb && !g_nRot)
{
    g_nGelb= 0;
    g_nRot= 1;
    GelbAus();
    RotEin();
}
else
{
}
```

© MicroConsult - Microelectronics Consulting & Training GmbH14.12.2015 F 5 MICROCONSULT

Was will uns dieser Code sagen?

→ Die Abfrage von vielen Statusvariablen deutet auf ein zustandsbasiertes System hin.

Es wurde versucht, eine einzelne Ampel mit ihren Phasen abzubilden.

```
if (g_nGelb == 1)
{
    g_nGruen = 0;
    g_nRot = 0;
    GelbEin();
}
else if (g_nRot == 1)
{
    g_nRot = 0;
    g_nGelb = 0;
    g_nGruen = 1;
    RotAus();
    GelbAus();
    GruenEin();
}
else if (g_nGruen == 1)
{
    g_nGruen = 0;
    g_nRot = 0;
    g_nGelb = 0;
    GruenAus();
    RotEin();
    GelbEin();
}
else
{
    g_nRot = 0;
    g_nGelb = 0;
    g_nGruen = 0;
    RotAus();
    GelbAus();
    GruenAus();
}
```



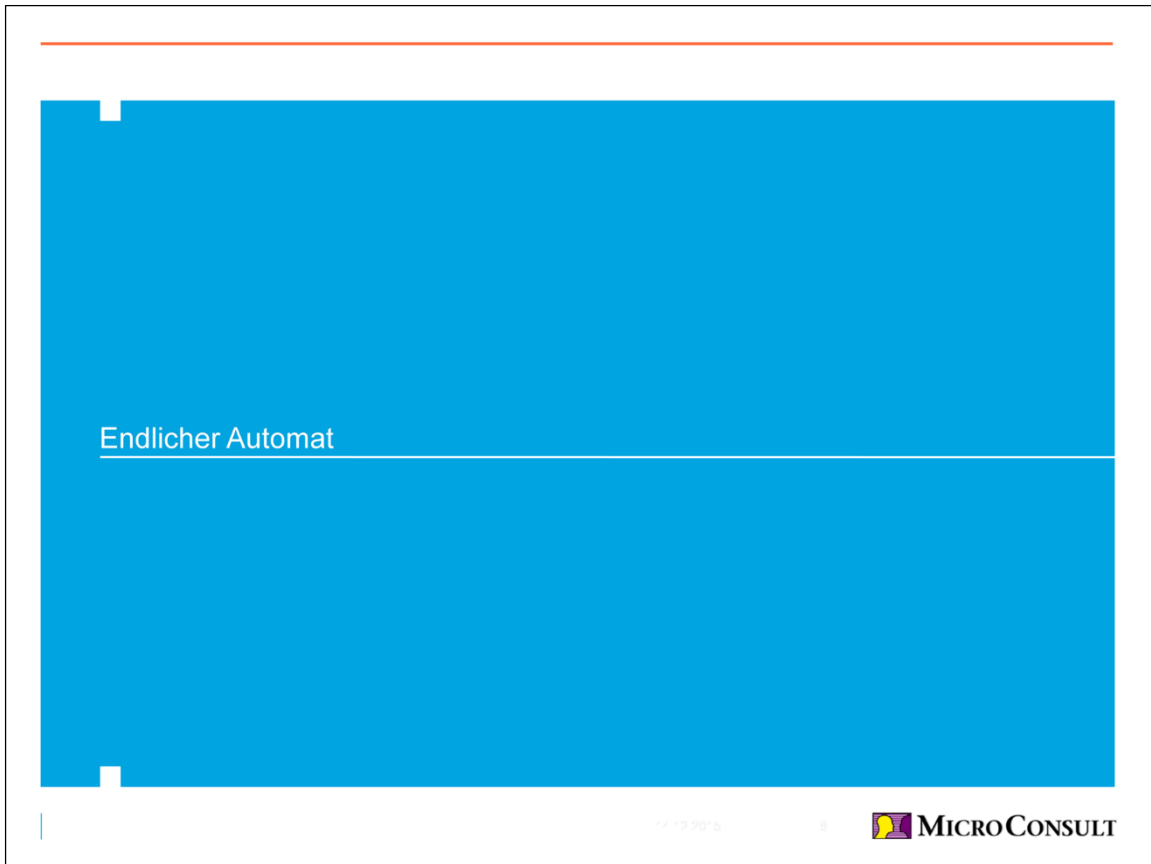
Wo ist das Problem des Beispiels?

Offensichtlich existiert versteckt im Code ein Zustandsautomat. Da er aber hinter vielen Variablen versteckt wird, schafft das einige Probleme:

- Der Automat (seine Logik) ist von außen nicht erkennbar.
- Eine Veränderung oder Erweiterung des Automaten ist nur mit großem Aufwand möglich.
- Es besteht die Gefahr, dass durch kleine Codeänderungen der ganze Automat unbrauchbar wird.

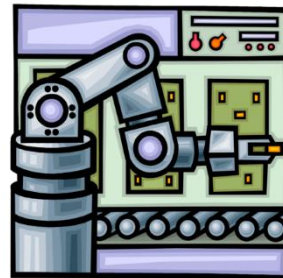
→ Machen Sie den Automaten sichtbar!

### 3 Endlicher Automat



Ein **endlicher Automat** (auch Zustandsmaschine, englisch *finite state machine (FSM)*)

- Modelliert ein Verhalten, das aus Zuständen, Zustandsübergängen und Aktionen besteht.
- Er heißt endlich, weil die Menge der Zustände, die er annehmen kann, endlich ist.



Ein **Zustand** zeigt den aktuellen Status an.

Ein **Zustandsübergang** (Transition) zeigt eine Änderung des Zustandes an.

Eine **Aktion** ist die Ausgabe des endlichen Automaten, die in einer bestimmten Situation erfolgt. Es gibt vier Typen von Aktionen:

- Eingangsaktion – wird beim Eintreten in einen Zustand ausgeführt.
- Ausgangsaktion – wird beim Verlassen eines Zustandes ausgeführt.
- Eingabeaktion – wird abhängig vom aktuellen Zustand und der Eingabe ausgeführt.
- Übergangsaktion – wird abhängig von einem Zustandsübergang ausgeführt.

## 3.1 Arten

Generell werden zwei Gruppen von endlichen Automaten unterschieden:

### **Akzeptoren**

Sie verarbeiten Eingabewerte und signalisieren durch ihren Zustand das Ergebnis nach außen. In der Regel werden Symbole (Buchstaben) als Eingabe benutzt.

Akzeptoren werden vorwiegend in der Wort- und Spracherkennung eingesetzt.

### **Transduktoren (Transducer)**

Transduktoren führen Aktionen in Abhängigkeit von Zustand und Eingabesignal aus. Sie werden vorwiegend für Steuerungsaufgaben eingesetzt.

Es werden grundsätzlich zwei Typen unterschieden:

- Moore-Automat
- Mealy-Automat

### 3.1.1 Moore-Automat

Endlicher Automat – Arten – Moore-Automat

Moore-Automat (nach dem Mathematiker Edward F. Moore (1925-2003))  
Ein Moore-Automat führt seine Aktionen in Abhängigkeit vom Zustand aus. Die Aktion wird ausgeführt, wenn der Zustand erreicht wird.

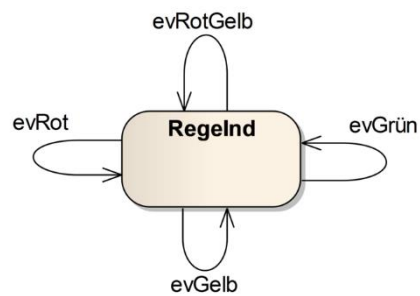
```
graph TD; Rot[Rot] -- evRot --> Gelb[Gelb]; Gelb -- evGelb --> Grün[Grün]; Grün -- evGrün --> RotGelb[RotGelb]; RotGelb -- evRotGelb --> Rot;
```

© MicroConsult - Microelectronics Consulting & Training GmbH 14.12.2015 F 12 MICROCONSULT

### 3.1.2 Mealy-Automat

Mealy-Automat (nach dem Mathematiker George H. Mealy (1927-2010))

Ein Mealy-Automat führt seine Aktionen in Abhängigkeit vom Zustandsübergang aus. Die Aktion wird ausgeführt, wenn der Zustandsübergang stattfindet.



## 3.2 Automaten in der Praxis

### Moore oder Mealy?

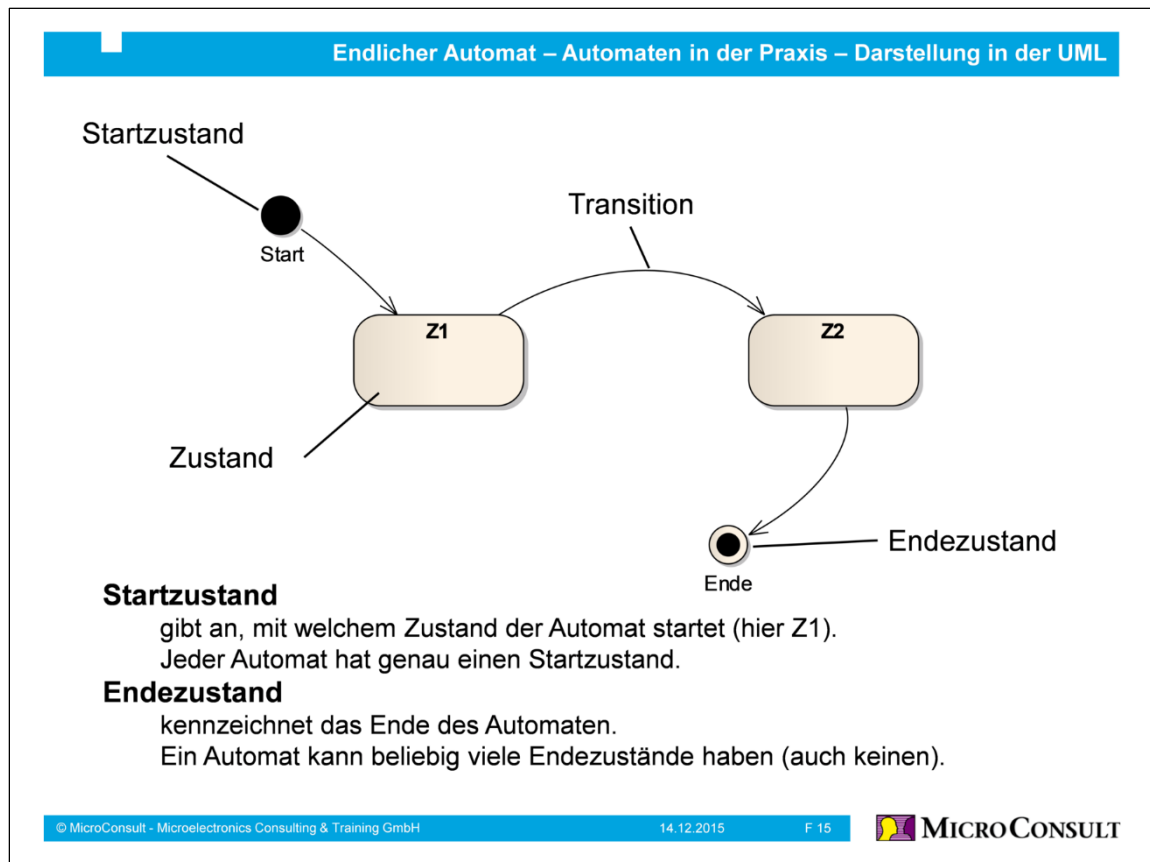
Welcher Automat sollte verwendet werden?

In der Praxis kommt meistens eine Mischform zum Einsatz, d.h. die Eigenschaften der Moore- und Mealy-Modelle werden kombiniert.

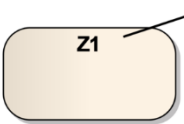
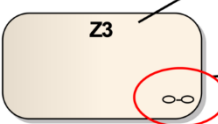
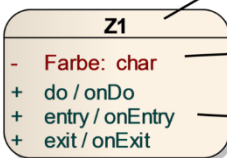
Gängige Modelle verwenden im Allgemeinen vier verschiedene Aktionen:


- Aktion an der Transition (Mealy)
- Aktion bei Erreichen des Zustands (Moore)
- Aktion im Zustand
- Aktion beim Verlassen des Zustandes

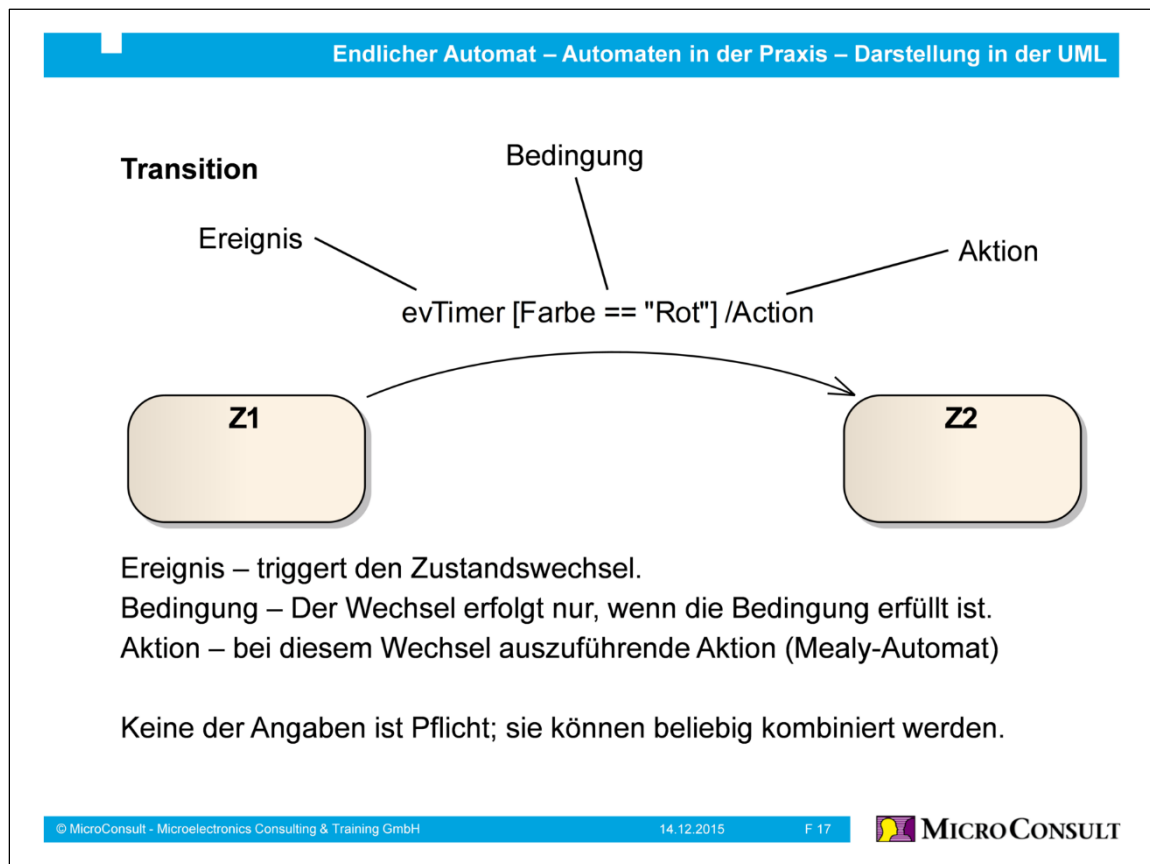
## 3.2.1 Darstellung in der UML



**Endlicher Automat – Automaten in der Praxis – Darstellung in der UML**

<p><b>Zustand</b> Einfache Darstellung</p>	 <p>Name</p>
<p>Zusammengesetzter Automat</p>	 <p>Name</p> <p>Hinweis auf eingebetteten Automaten</p>
<p>Ausführliche Darstellung</p>	 <p>Name</p> <p>Zustandsvariable</p> <p>Aktionen</p>

© MicroConsult - Microelectronics Consulting & Training GmbH      14.12.2015      F 16      



Abhängig von den Randbedingungen des Projekts werden nur Ereignisse oder nur Bedingungen oder eine Kombination aus beiden verwendet.

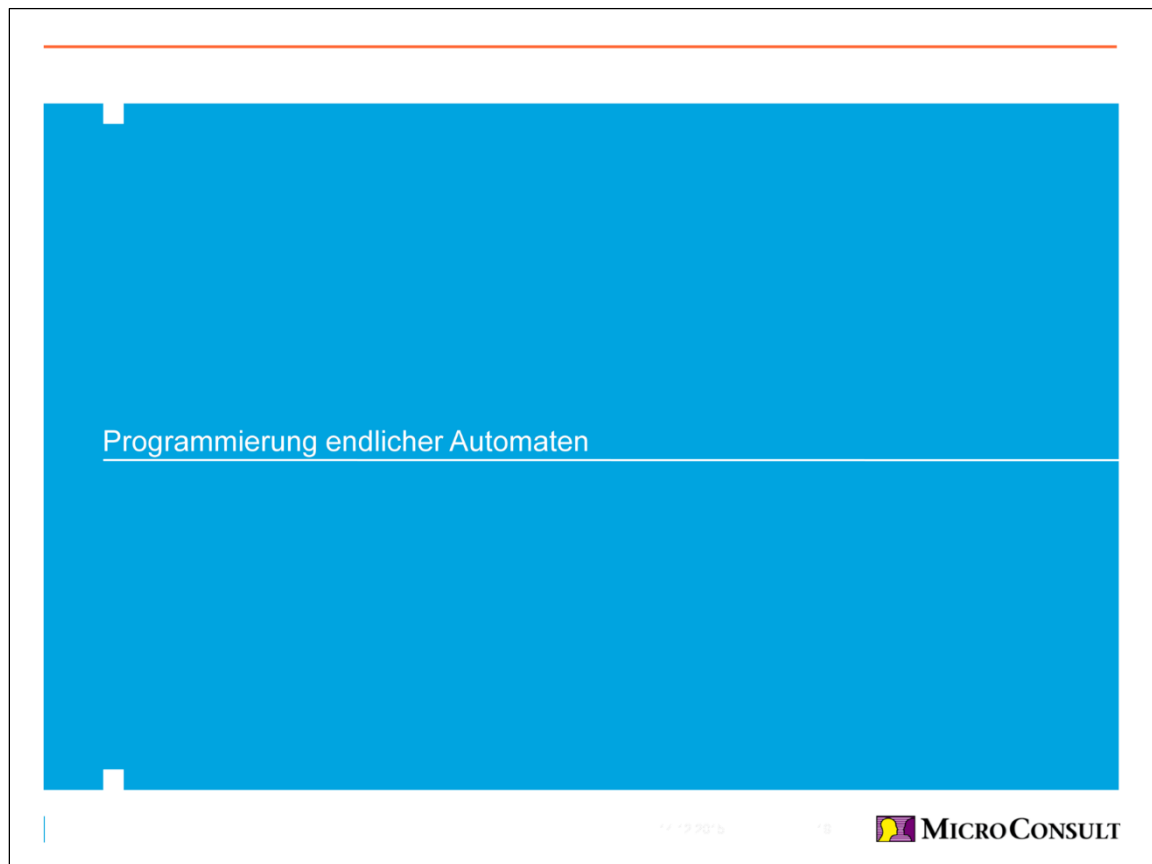
### 3.2.2 Design

Das Design eines Automaten sollte immer dokumentiert werden und nicht nur im Kopf des Entwicklers stattfinden.

- Dafür reicht ein einfaches Zeichentool.
- Ein UML-Tool ist empfehlenswert, weil auch andere Entwurfsschritte gleich mit dokumentiert werden können.

Alternativ oder ergänzend zum graphischen Design kann auch eine Zustandstabelle eingesetzt werden.

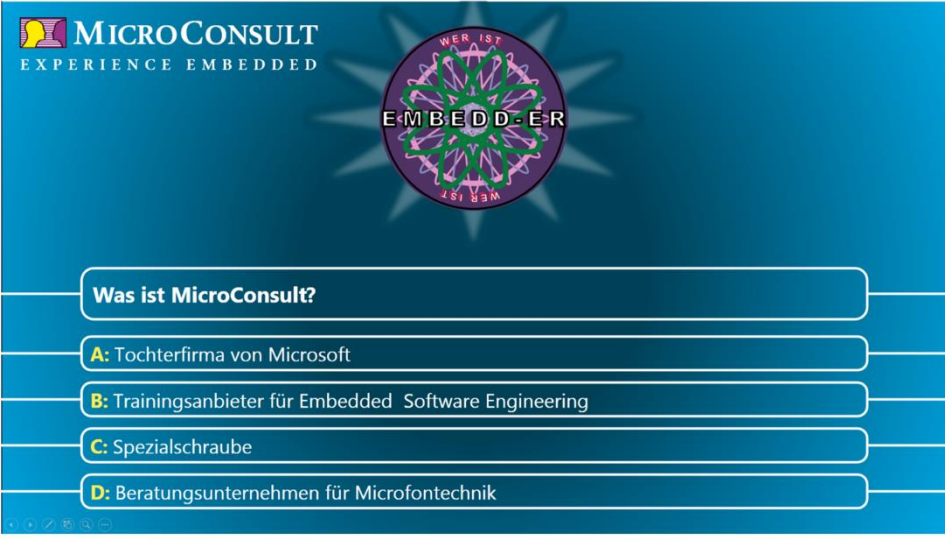
## 4 Programmierung endlicher Automaten



## 4.1 Der Automat

Programmierung endlicher Automaten – Der Automat

Als Grundlage dient der Zustandsautomat, der für die Steuerung einer einfachen Quiz-App verwendet wird.



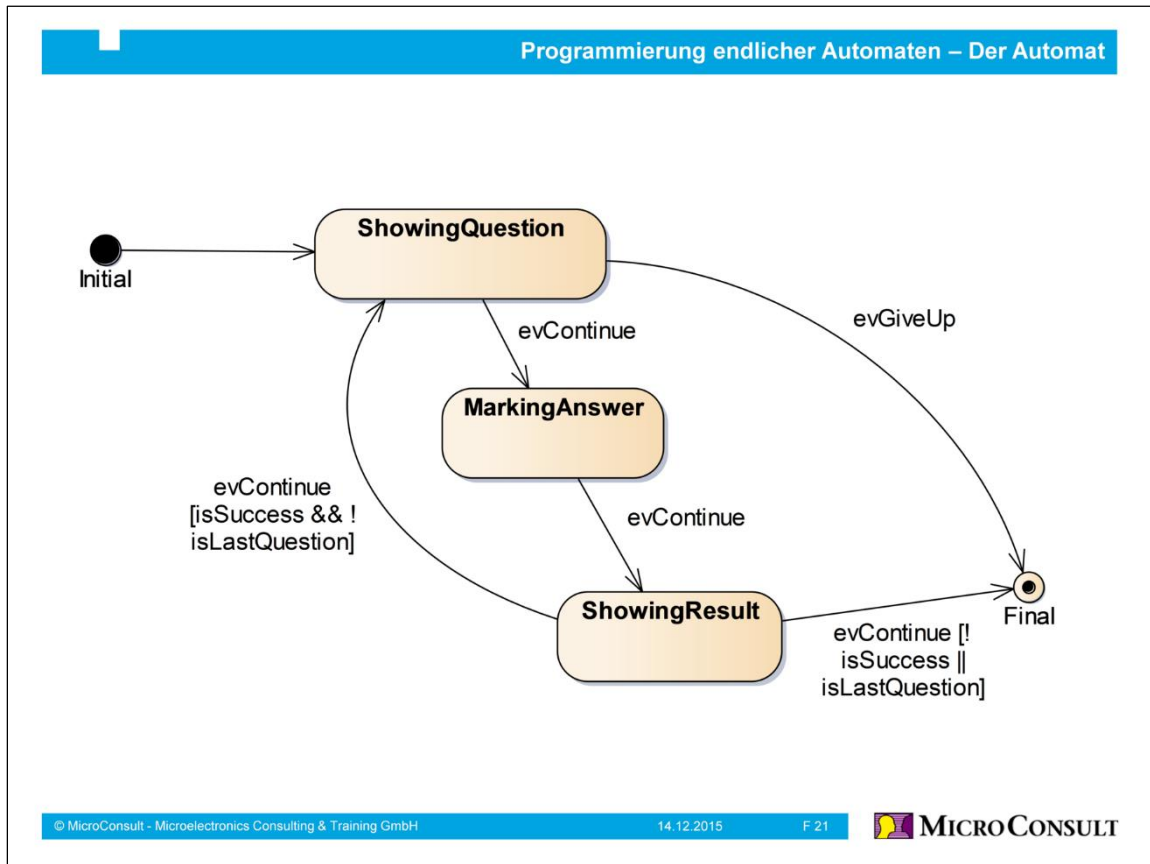
**MICROCONSULT**  
EXPERIENCE EMBEDDED

WER IST EMBEDD-ER

**Was ist MicroConsult?**

- A:** Tochterfirma von Microsoft
- B:** Trainingsanbieter für Embedded Software Engineering
- C:** Spezialschraube
- D:** Beratungsunternehmen für Microfontechnik

© MicroConsult - Microelectronics Consulting & Training GmbH 14.12.2015 F 20 **MICROCONSULT**



## 4.2 Realisierung

Ein Automat stellt den Zustand eines Objektes dar, d.h. der Automat ist Bestandteil einer Klasse.

<b>QuestionManager</b>	
-	<b>m_currentState: States</b>
-	EnterMarkingAnswer(char): void
-	EnterShowingQuestion(char): void
-	EnterShowingResult(char): void
+	ProcessEvent(Events, char): int
+	Start(): void

### 4.2.1 switch-case

Programmierung endlicher Automaten – Realisierung – switch-case

Die klassische switch-case-Implementierung ist erst einmal ganz einfach umzusetzen.

Ereignis

switch

case ev1:

case ev2:

case ev3:

Aktueller Zustand

switch

case Z1:

case Z2:

case Z3:

Bedingung

if(*Bedingung*)

setze Folgezustand  
Aktion()

**QuestionManager**  
 - m\_currentState: States  
 - EnterMarkingAnswer(char): void  
 - EnterShowingQuestion(char): void  
 - EnterShowingResult(char): void  
 + ProcessEvent(Events, char): int  
 + Start(): void

© MicroConsult - Microelectronics Consulting & Training GmbH
14.12.2015
F 23

### Programmierung endlicher Automaten – Realisierung – switch-case

#### Vorteile

- einfaches Prinzip

#### Nachteile

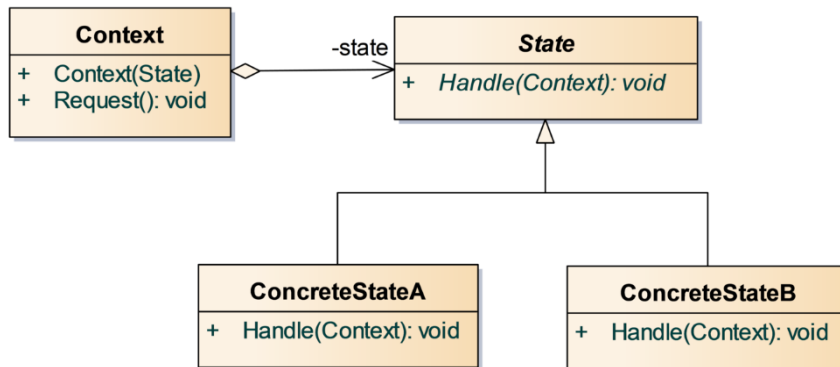
- Die Mechanik des Automaten wird mit der Funktionalität der Applikation verwoben.
  - Was gehört zum Automaten?
  - Was ist Funktionalität?
- Wird ein neuer Automat in die Applikation eingefügt, muss alles (auch der Automat selbst) komplett neu geschrieben werden.
- Schwer zu überblicken
- Hoher Testaufwand

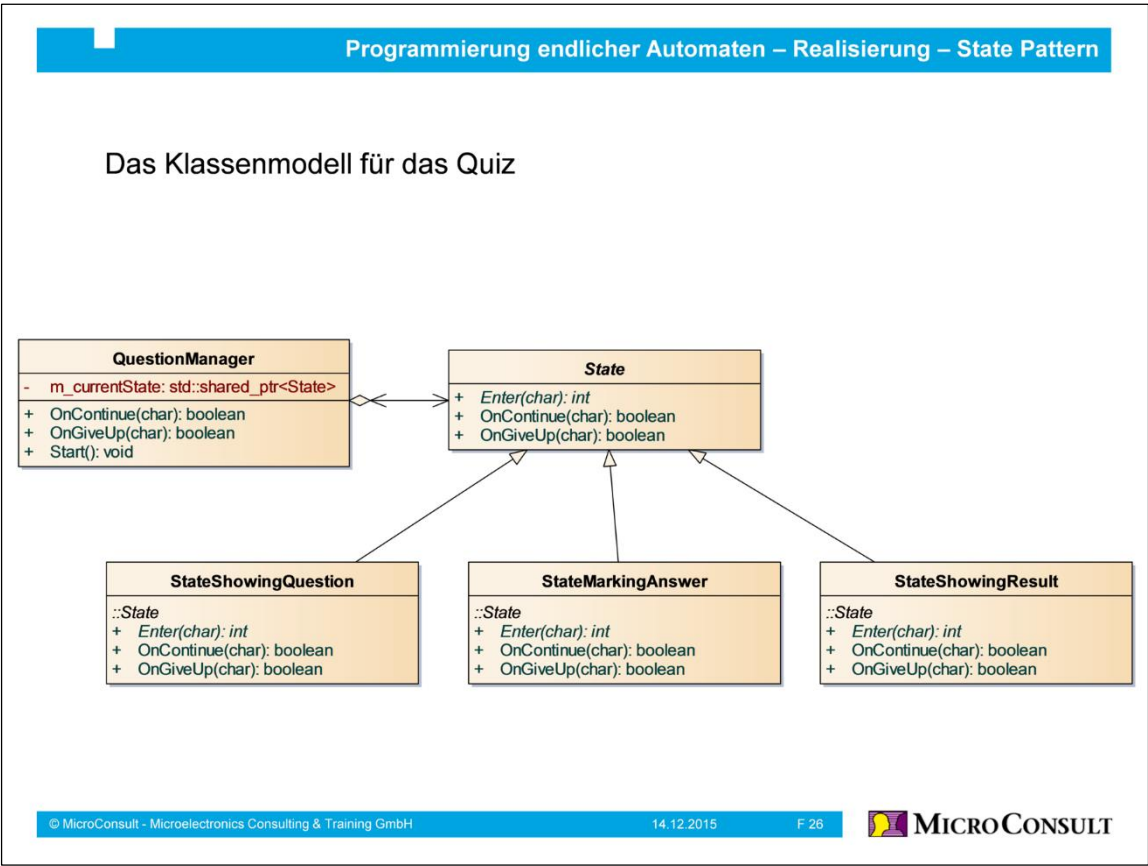
## 4.2.2 State Pattern

Programmierung endlicher Automaten – Realisierung – State Pattern

Das State Pattern wurde von der Gang of Four (GoF) um Erich Gamma vorgestellt.

Hier wird die Objektorientierung etwas besser ausgenutzt als bei der einfachen switch-case-Implementierung.





Programmierung endlicher Automaten – Realisierung – State Pattern

Vorteile

- Objektorientierter Ansatz
- Einfache Erweiterung, ein Zustand kann sehr leicht hinzugefügt werden
- Ausnutzung der Vererbung bei gleicher Funktionalität in mehreren Zuständen möglich

Nachteile

- Eventuell große Anzahl von Klassen – eine pro Zustand
- Die Mechanik des Automaten ist immer noch nicht klar von der Funktionalität der Applikation getrennt

### 4.2.3 Tabelle

Programmierung endlicher Automaten – Realisierung – Tabelle

Günstig ist es, Automat und Funktionalität voneinander zu trennen.

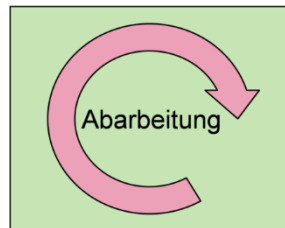
- Die Mechanik des Automaten wird nur einmal programmiert und immer wiederverwendet.
- Der Automatencode ist separat testbar.

```
graph LR; A[Funktionalität] --> B[Parametrierung]; B --> C[Abarbeitung];
```

Bei einem tabellengesteuerten Automaten wird die Mechanik des Automaten klar von der Funktion der Anwendung getrennt.

© MicroConsult - Microelectronics Consulting & Training GmbH 14.12.2015 F 28 MICROCONSULT

Programmierung endlicher Automaten – Realisierung – Tabelle



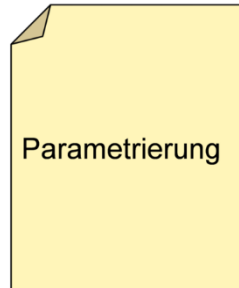
Mechanik des Automaten

→ Funktion zur Abarbeitung der Tabelle

- durchsucht die Tabelle nach einem passenden Eintrag
- arbeitet alle Anweisungen des gefundenen Eintrags ab

Funktionalität

- Funktionalität, die der Automat steuern soll  
→ Funktionen des eigentlichen Programmes
- auszuführende Aktionen
  - zu prüfende Bedingungen

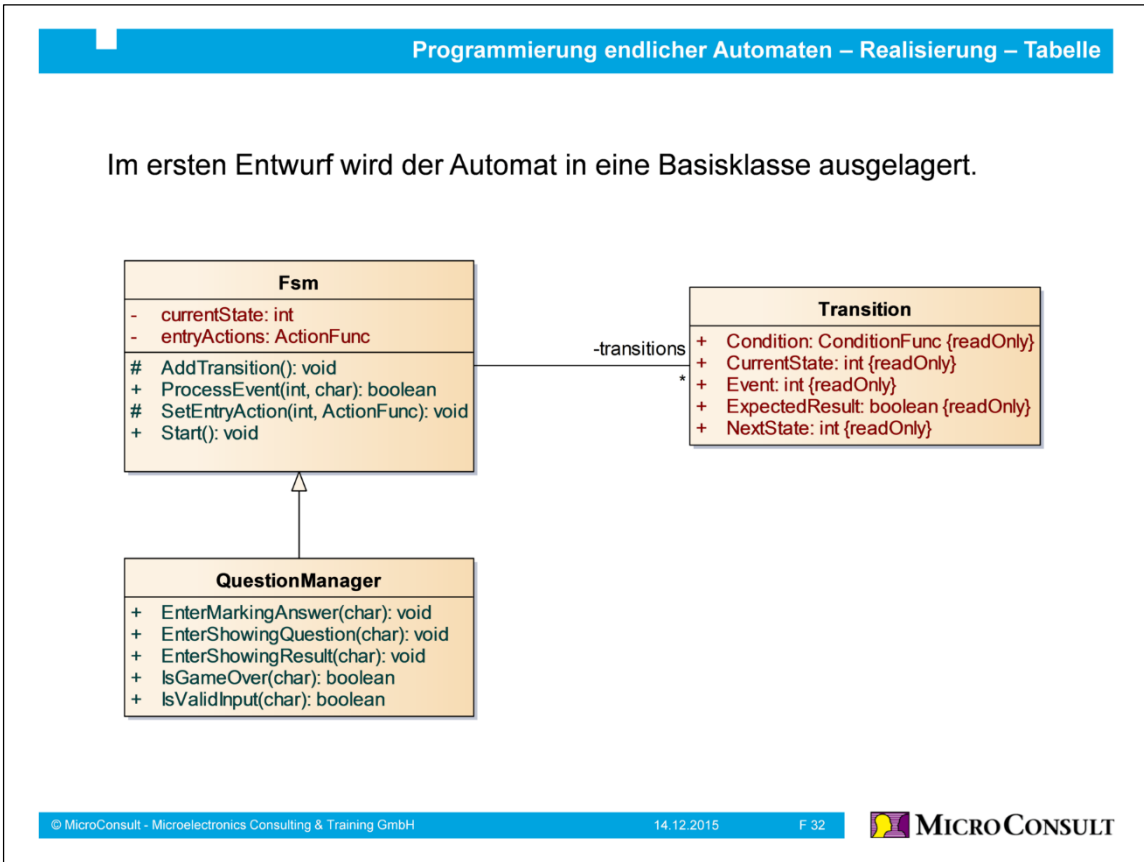


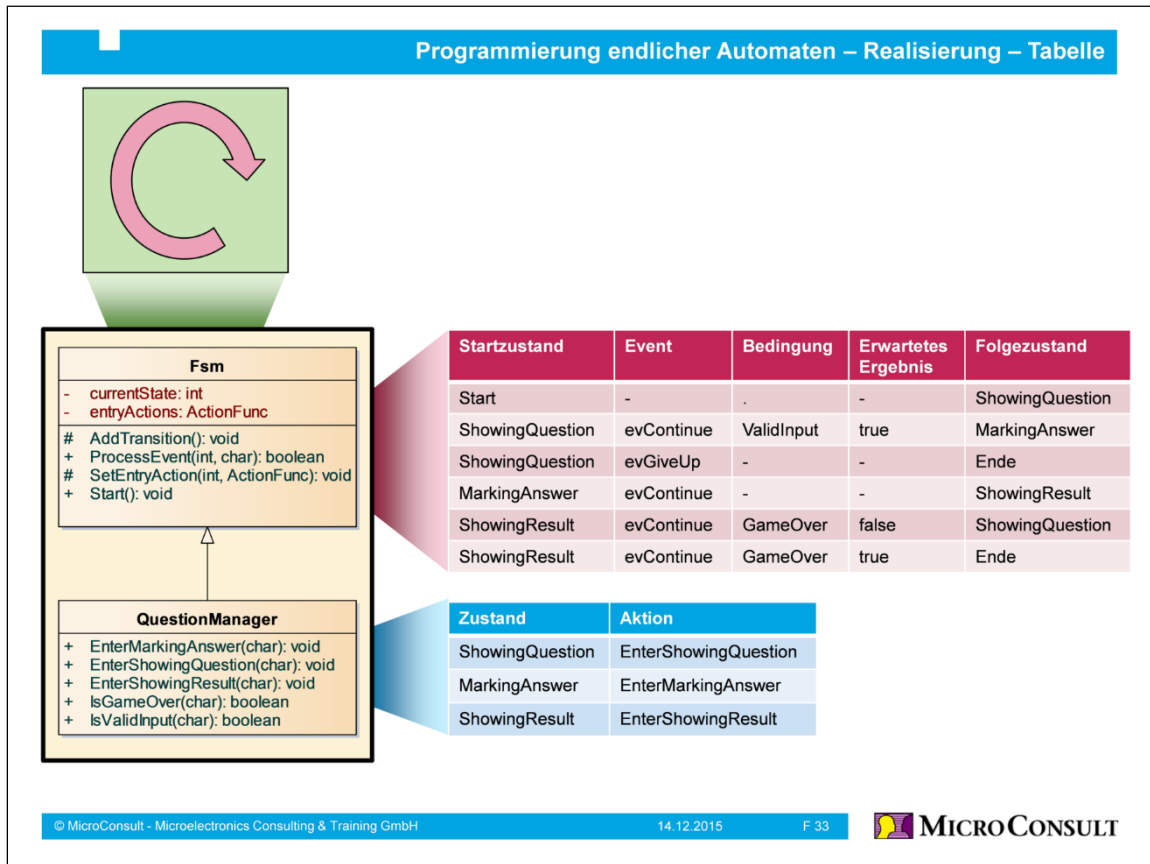
Beschreibung des Automaten

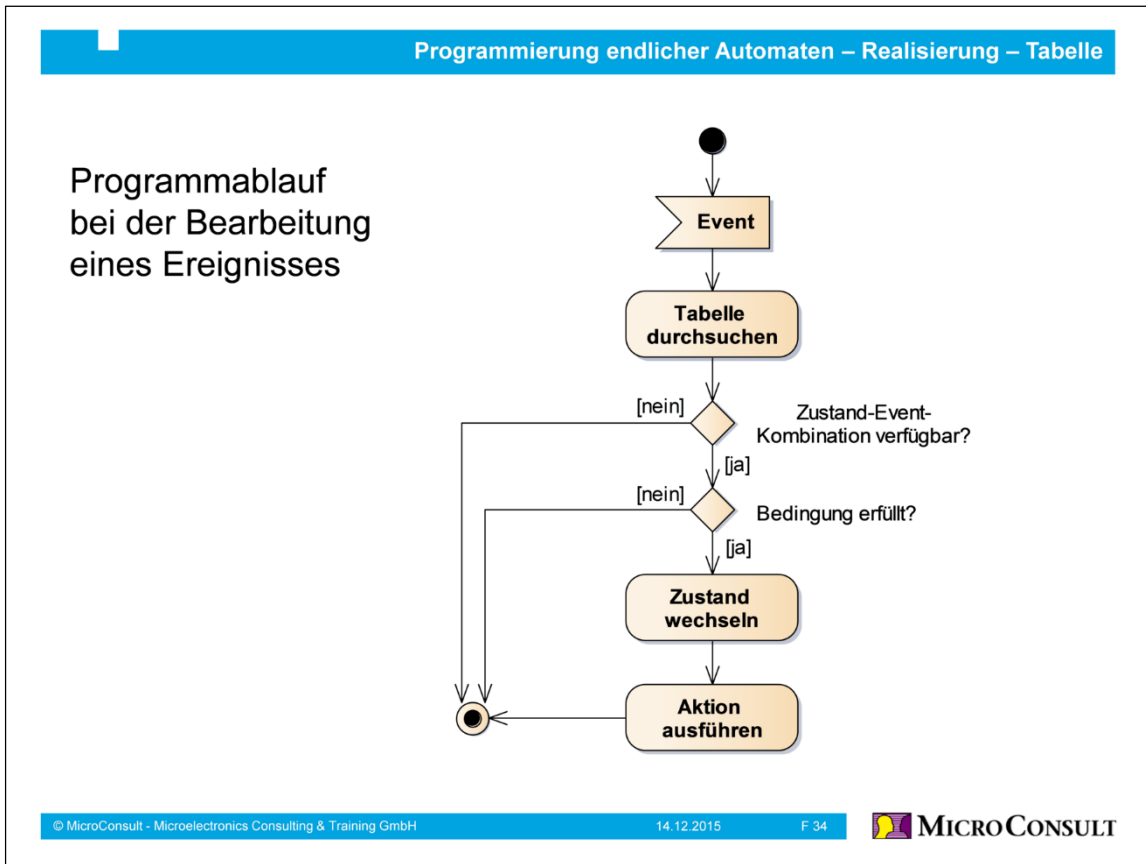
→ Zustand-Ereignis-Tabelle

- enthält die Beschreibungen für alle Transitionen des Automaten

4.2.3.1 Automat als Basisklasse







Programmierung endlicher Automaten – Realisierung – Tabelle

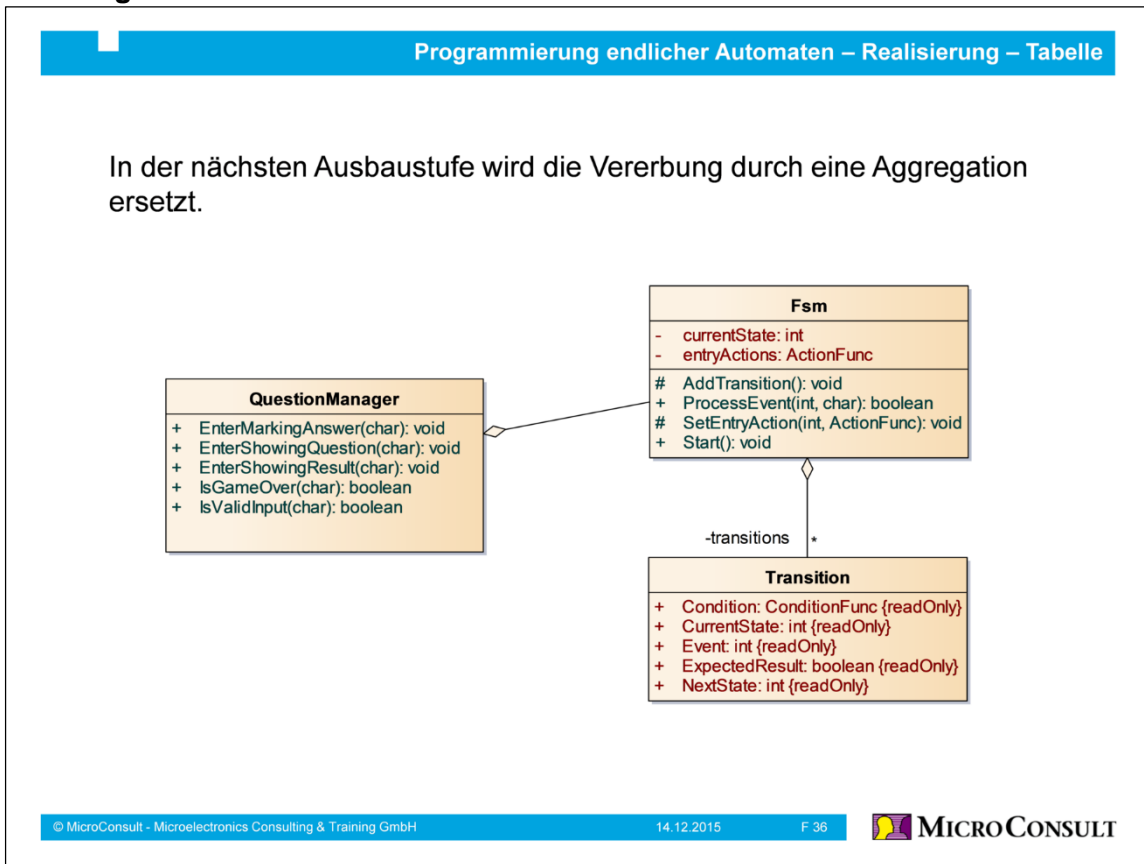
Vorteile:

- Klare Trennung von Automat und Verhalten
- Einfaches Prinzip
- Direkter Bezug von Transition im Modell des Automaten zu dem entsprechenden Tabelleneintrag

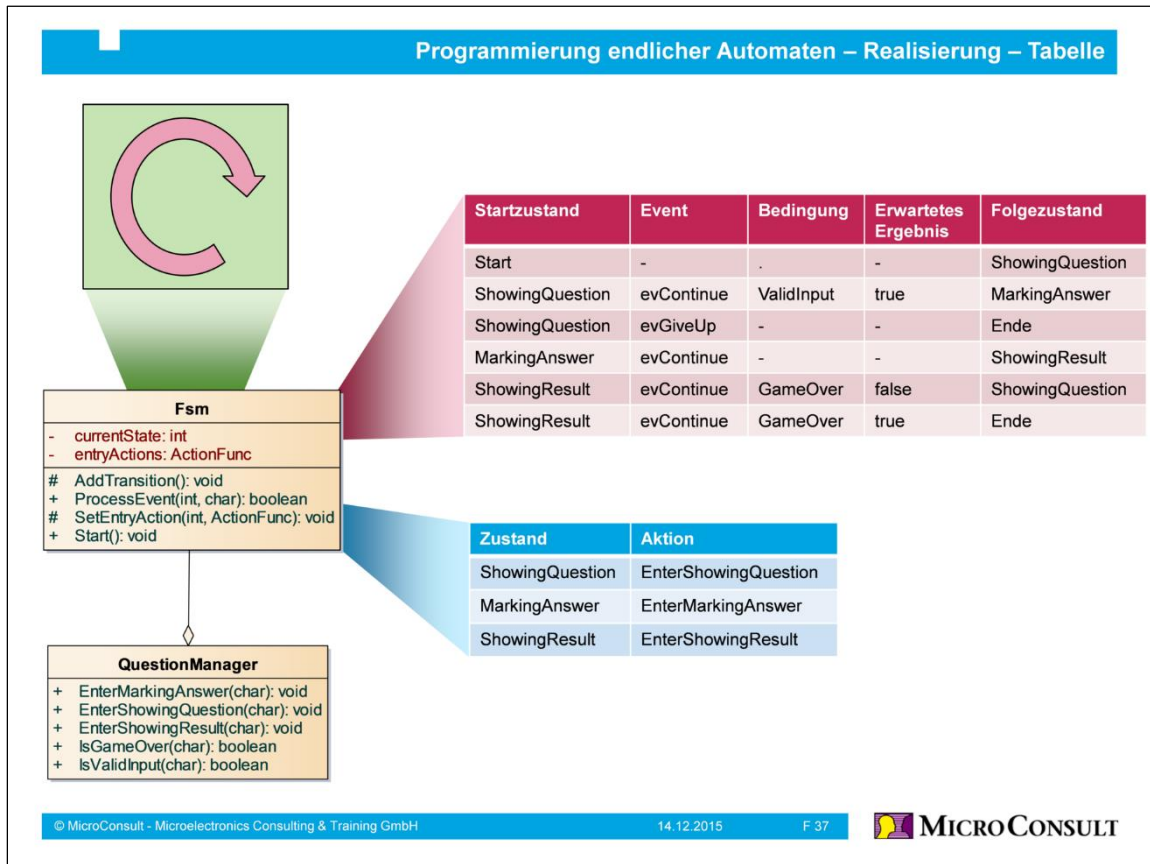
Nachteile:

- Im schlimmsten Fall müssen alle Tabelleneinträge durchsucht werden.
- Durch die Vererbung gibt es eine starke Abhängigkeit zwischen Automat und Realisierung.
- Nur Memberfunktionen können als Bedingung oder Aktion verwendet werden.

### 4.2.3.2 Eingebetteter Automat



Es muss nicht zwingend mit Aggregation gearbeitet werden. Durch die klare Trennung der Klassen kann das Automatenobjekt auch außerhalb der benutzenden Klasse erzeugt werden. Damit würde die Kopplung noch loser.



**Programmierung endlicher Automaten – Realisierung – Tabelle**

Der Programmablauf bei der Bearbeitung eines Ereignisses unterscheidet sich nicht von der Vererbungs-Variante

```
graph TD; Start(( )) --> Event[/Event/]; Event --> Tabelle[Tabelle durchsuchen]; Tabelle --> D1{Zustand-Event-Kombination verfügbar?}; D1 -- ja --> D2{Bedingung erfüllt?}; D1 -- nein --> Start; D2 -- ja --> Zustand[Zustand wechseln]; D2 -- nein --> Start; Zustand --> Aktion[Aktion ausführen]; Aktion --> Start;
```

Programmierung endlicher Automaten – Realisierung – Tabelle

Bei der Variante "FSM als Basisklasse" können nur Memberfunktionen als Bedingungs- oder Aktionsfunktion verwendet werden. Es ist sogar noch ein (unschöner) cast notwendig, um den Code übersetzen zu können.

```
AddTransition(
    static_cast<int>(States::ShowingQuestion), // Startzustand
    static_cast<int>(Events::evContinue), // Event
    static_cast<ConditionFunc>(&QuestionManager::IsValidInput), // Bedingung
    true, // erwartetes Ergebnis
    static_cast<int>(States::MarkingAnswer)); // Folgezustand
```

In der neuen Variante soll es nun möglich sein, beliebige Funktionen zu verwenden.

In der Transition wird der Funktionspointer auf eine Memberfunktion durch ein `std::function`-Objekt ersetzt.

```
using ConditionFunc = bool (Fsm::*)(char) const;  
using ActionFunc = void (Fsm::*)(char);
```



```
using ConditionFunc = std::function<bool(char)>;  
using ActionFunc = std::function<void(char)>;
```

Dadurch kann in einer Transition eine beliebige Funktion mit der angegebenen Signatur gespeichert werden.

Programmierung endlicher Automaten – Realisierung – Tabelle

Normale Funktion:

```
m_fsmQuestions.AddTransition(
    static_cast<int>(States::ShowingQuestion),
    static_cast<int>(Events::evContinue),
    ::IsValidInput,
    true,
    static_cast<int>(States::MarkingAnswer));
```

Jetzt können auch Lambda-Ausdrücke verwendet werden:

```
m_fsmQuestions.AddTransition(
    static_cast<int>(States::ShowingQuestion),
    static_cast<int>(Events::evContinue),
    [](char data) { return data >= 'a' && data <= 'd'; },
    true,
    static_cast<int>(States::MarkingAnswer));
```

Wenn allerdings eine Memberfunktion angegeben wird, kommt es zu einem Compilerfehler (Failed to specialize function template 'unknown-type std::invoke(\_Callable &&, \_Types &&...)' ):

```
m_fsmQuestions.AddTransition(  
    static_cast<int>(States::ShowingQuestion),  
    static_cast<int>(Events::evContinue),  
    &QuestionManager::IsValidInput,  
    true,  
    static_cast<int>(States::MarkingAnswer));
```

Das liegt am `this`-Pointer, der implizit übergeben wird (die Memberfunktion hat einen Parameter mehr).

Dadurch passt die Signatur der Memberfunktion (2 Parameter) nicht zur verlangten Signatur (1 Parameter).

Programmierung endlicher Automaten – Realisierung – Tabelle

Die Standardbibliothek hat eine Lösung: `std::bind()`.

Der `this`-Pointer wird fest an den Aufruf gebunden. Der zweite Parameter wird durchgereicht. Dadurch wird die vorhandene Signatur an die geforderte angepasst.

```
m_fsmQuestions.AddTransition(
    static_cast<int>(States::ShowingQuestion),
    static_cast<int>(Events::evContinue),
    bind(&QuestionManager::IsValidInput, this, _1),
    true,
    static_cast<int>(States::MarkingAnswer));
```

### Programmierung endlicher Automaten – Realisierung – Tabelle

#### Vorteile:

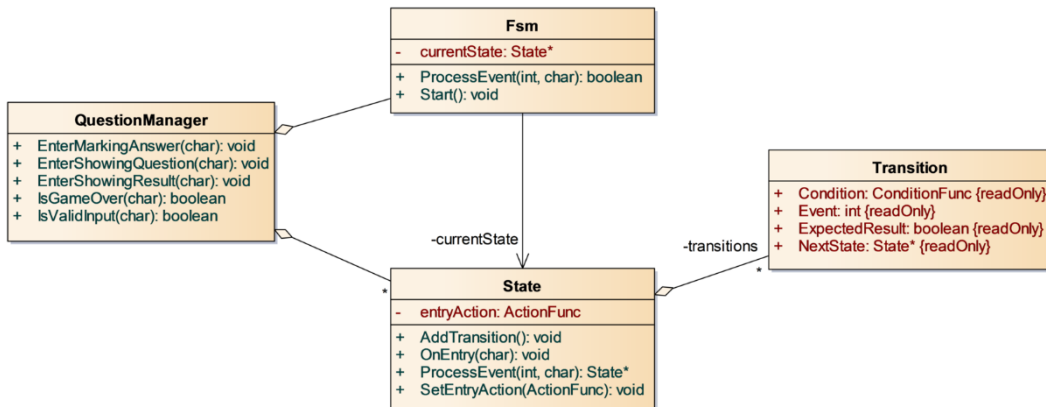
- Klare Trennung von Automat und Verhalten
- Einfaches Prinzip
- Direkter Bezug von Transition im Modell des Automaten zu dem entsprechenden Tabelleneintrag
- Lose Kopplung zwischen Automat und Realisierung
- Beliebige Funktionen können als Bedingung oder Aktion verwendet werden.

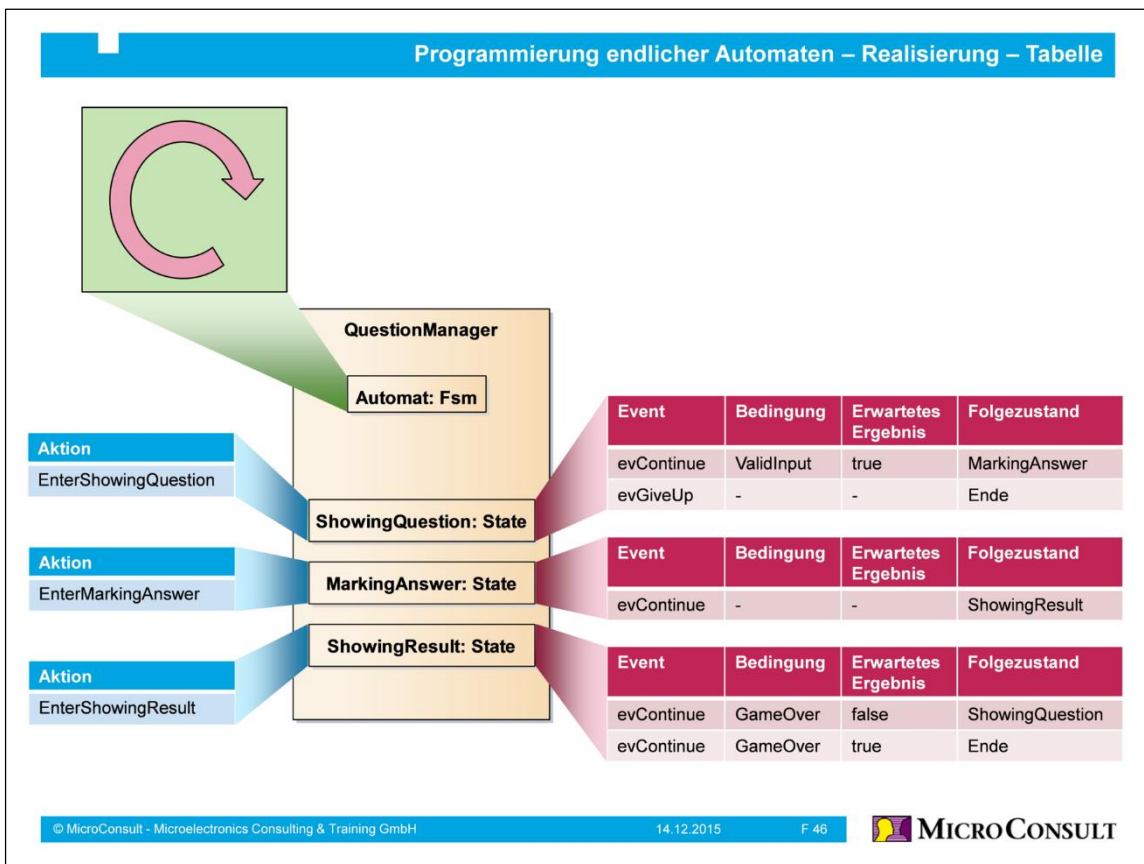
#### Nachteile:

- Im schlimmsten Fall müssen alle Tabelleneinträge durchsucht werden.

Programmierung endlicher Automaten – Realisierung – Tabelle

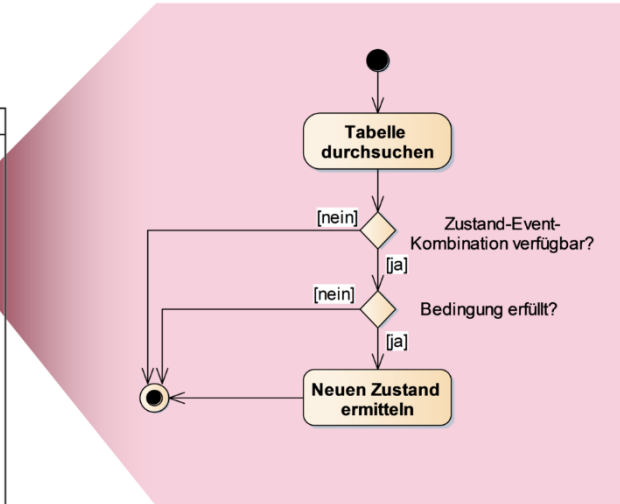
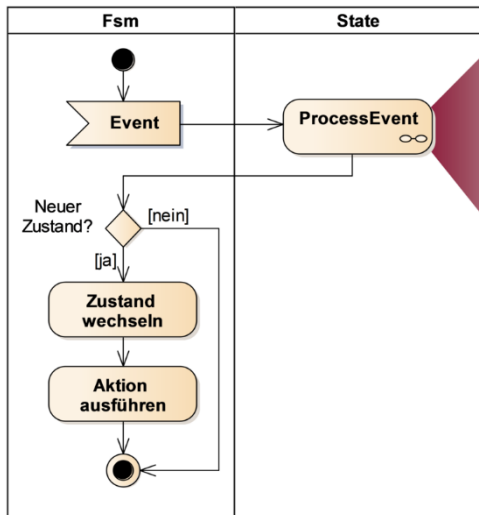
Um die große Tabelle im Automaten zu ersetzen, dient das State Pattern als Inspiration. Der Zustand wird zur Klasse. Allerdings werden die Zustände eines Automaten nicht als Klasse, sondern als Objekt abgebildet.





Programmierung endlicher Automaten – Realisierung – Tabelle

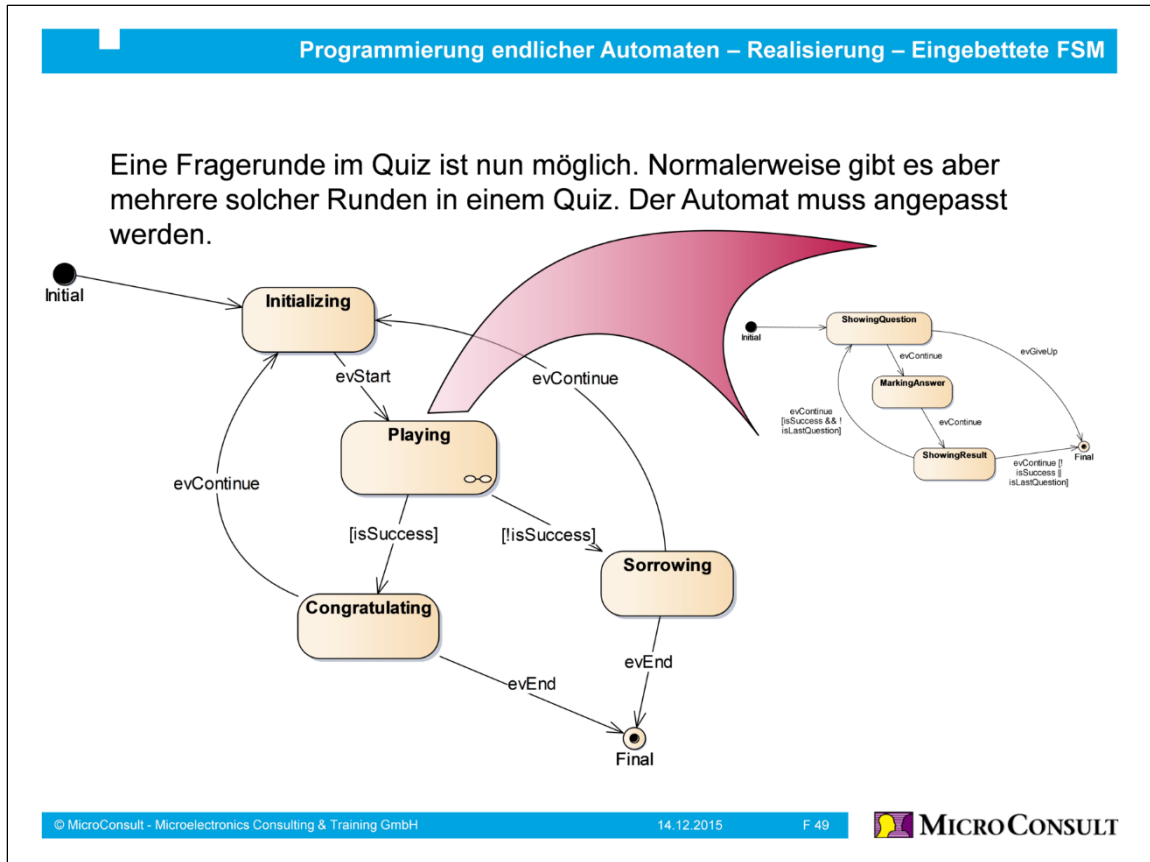
In dieser Variante verteilt sich die Bearbeitung des Events über zwei Klassen



#### Vorteile:

- Klare Trennung von Automat und Verhalten
- Einfaches Prinzip (die komplexere Struktur ist von außen nicht sichtbar)
- Direkter Bezug von Transition im Modell des Automaten zu dem entsprechenden Tabelleneintrag
- Lose Kopplung zwischen Automat und Realisierung
- Beliebige Funktionen können als Bedingung oder Aktion verwendet werden.
  
- Es müssen nur noch sehr kurze Tabellen durchsucht werden.

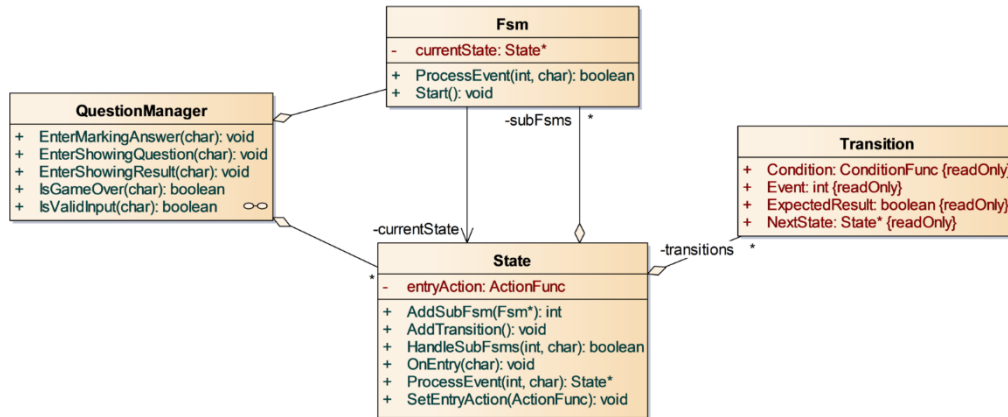
### 4.2.4 Eingebettete FSM



## Programmierung endlicher Automaten – Realisierung – Eingebettete FSM

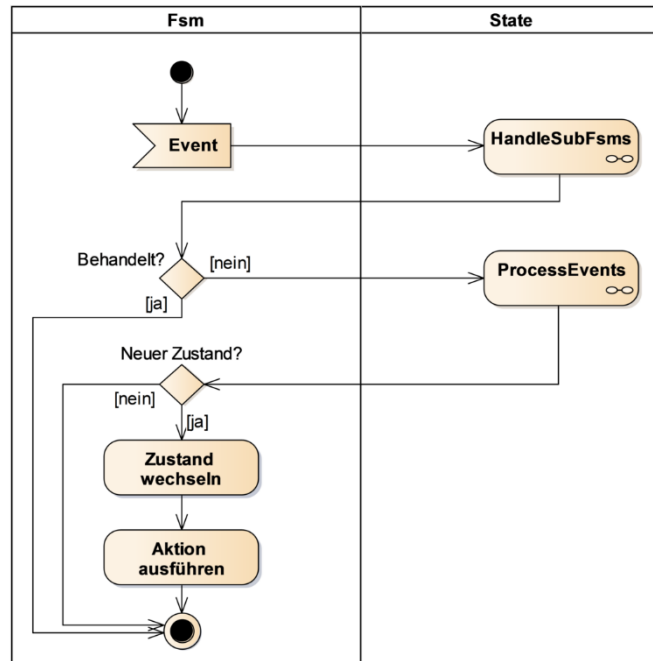
Auf den ersten Blick hat sich nichts geändert. Die meisten Änderungen sind intern. Lediglich die Klasse State bekommt drei neue Member:

- subFsms
- AddSubFsm ()
- HandleSubFsms ()



Programmierung endlicher Automaten – Realisierung – Eingebettete FSM

Bevor der Zustand selbst das Ereignis bearbeitet, wird geprüft, ob ein untergeordneter Automat für dieses Ereignis zuständig ist.



**Programmierung endlicher Automaten – Realisierung – Eingebettete FSM**

Die Klasse `State` übernimmt die Kontrolle der untergeordneten Automaten.

HandleSubFsms

ProcessEvent

© MicroConsult - Microelectronics Consulting & Training GmbH
14.12.2015
F 52
 MICROCONSULT

## 4.3 Ausblick

### Mögliche Erweiterungen:

- Um die Wiederverwendbarkeit zu erhöhen, ist es sinnvoll, die Automatenklassen als Template zu gestalten. Dann können beliebige Daten an die Methode `ProcessEvent()` übergeben werden.
- Umwandlung der Ereignisse von `Enum`-Werten (`int`) in Klassen
- Asynchrone Abarbeitung der Ereignisse
  - Die Methode `ProcessEvent()` läuft in einem separaten Thread.
  - Ankommende Ereignisse werden in einer Queue zwischengespeichert.
- Weitere Erhöhung des Komforts
  - Zustandsobjekte werden vom Automaten selbstständig erzeugt und z.B. nur über selbstgewählte Namen angesprochen.

#### Alternativen:

- Nutzung der Boost-Bibliothek. Dort werden zwei verschiedene Automatenimplementierungen angeboten.
- Generierung der Automaten aus Modellierungstools (z.B. UML-Tool) heraus

## 5 Fazit

Fazit

Es lohnt sich, Automaten im Code zu identifizieren und formal zu designen.

Zu viele Zustände machen Probleme – besser ist es, eine Hierarchie aufzubauen.

Mit der Programmiersprache C++ können moderne objektorientierte Automaten programmiert werden.

Je nach den Randbedingungen des Projektes kann der Einsatz von speziellen Werkzeugen hilfreich sein.


## 6 Download-Link

Download-Link

Im Internet sind Beispiele für den Vortrag als Download unter folgender Adresse verfügbar:

<http://download.microconsult.net/fl/ese-2015/QuizChampion.zip>

Bitte beachten Sie, dass der komplette Pfad inklusive Dateinamen angegeben wird.

© MicroConsult - Microelectronics Consulting & Training GmbH 14.12.2015 F 56  MICROCONSULT