Entwurf und Implementierung von Zustandsautomaten

Frank Listing
f.listing@microconsult.com

MICRO CONSULT

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?

Welche Werkzeuge helfen bei der Entwicklung von Automaten?

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

Die übliche Fachliteratur ignoriert heutzutage gerne die Programmiersprache C.

Modellierungstools mit Codegenerierung sind oft zu teuer oder für den benutzten Mikrocontroller nicht einsetzbar.

Einführung

Aufgabe:

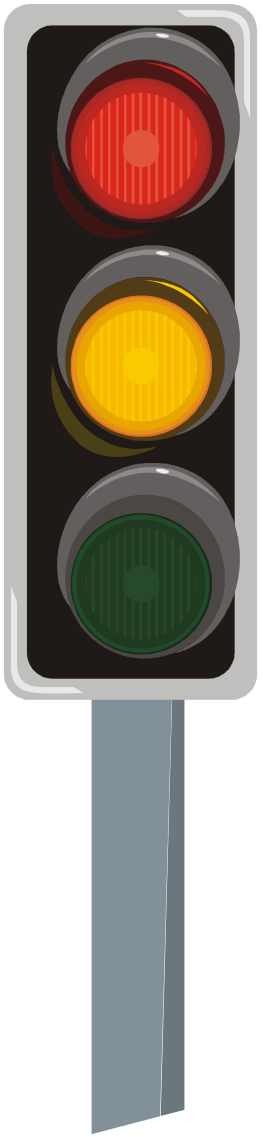
Steuern Sie die Lampen einer Ampel an.

Die Umschaltung erfolgt immer nachdem ein Zeitimpuls eingetroffen ist.

Ampelphasen:

- Rot
- Rot-Gelb
- Grün
- Gelb

Nach dem Einschalten soll die Ampel rot leuchten.



Das Problem mal schnell gelöst:

```
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
    {
    }
}
```

Welche Lehre kann aus dem Beispiel gezogen werden?

Die VHIT-Methode (vom Hirn ins Terminal) ist nicht die beste Lösung.

Besser ist es, sich erst einmal ein paar Gedanken zu machen:

- Wie ist das System einzuordnen?
- Wie kann das System modelliert werden?
- Wie wird das Modell in Code umgesetzt?

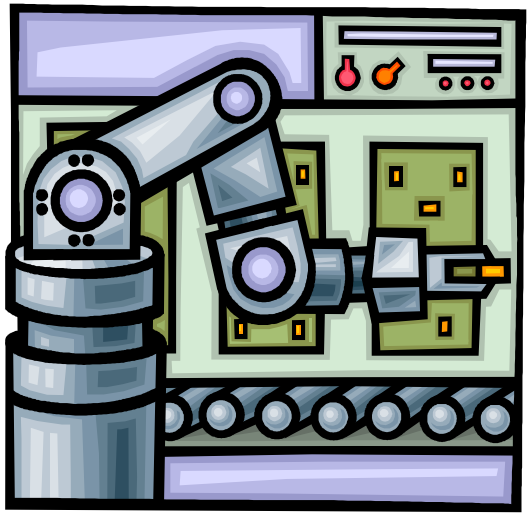
Offensichtlich handelt es sich hier um ein zustandsbasiertes System.

→ Einsatz eines Zustandsautomaten

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.



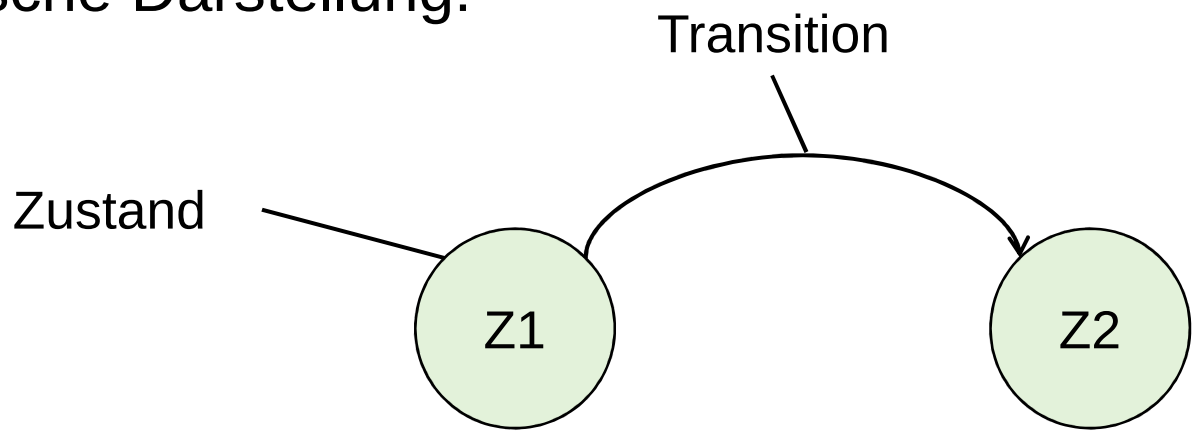
Ein **Zustand** zeigt den aktuellen Status an.

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

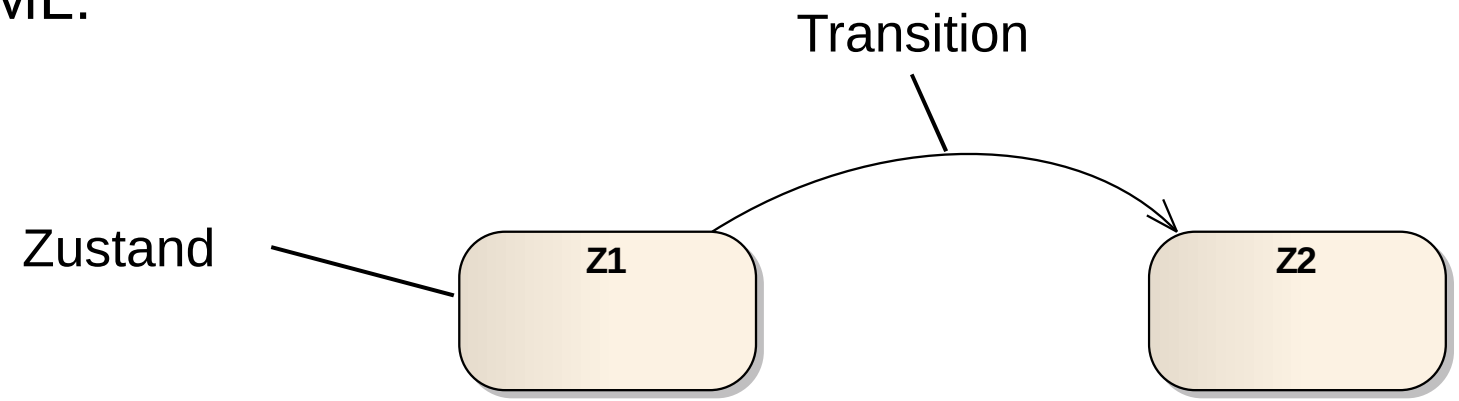
Eine **Aktion** ist die Ausgabe des endlichen Automaten, die in einer bestimmten Situation erfolgt. Es gibt vier Typen von Aktionen

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

Klassische Darstellung:



UML:



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)

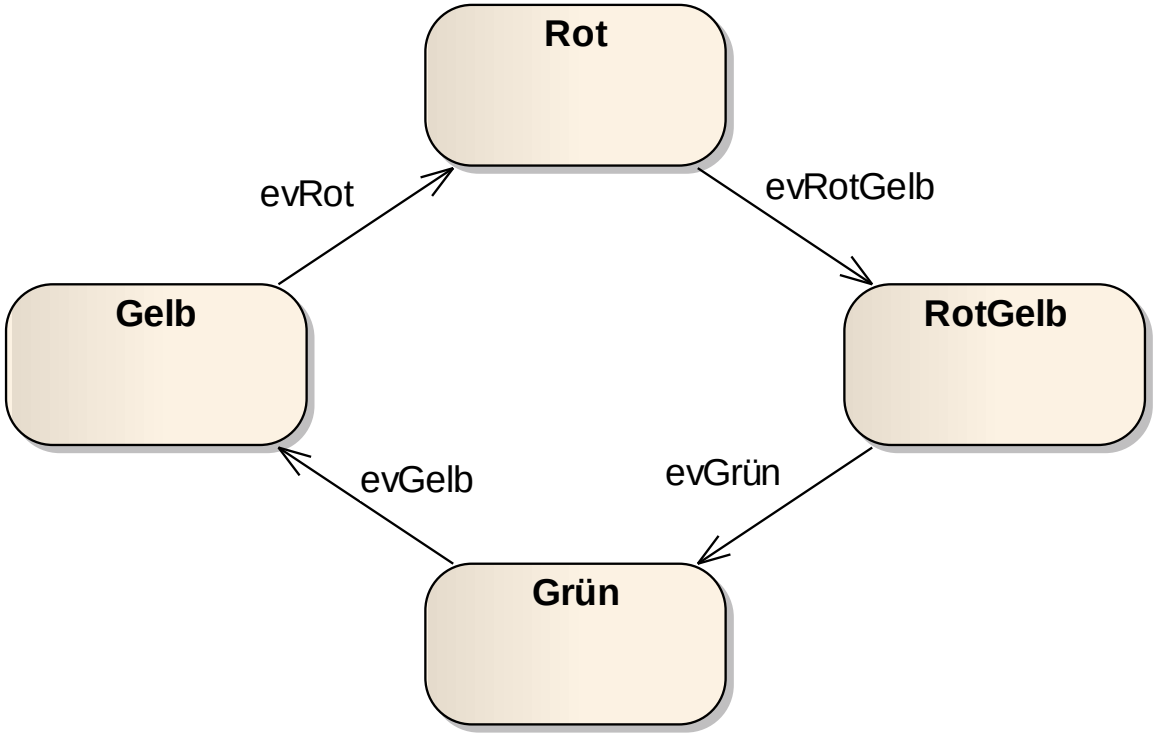
Transduktoren führen Aktionen in Abhängigkeit von Zustand und Eingabesignal aus. Sie werden vorwiegend für Steuerungsaufgaben eingesetzt.

Es werden grundsätzlich zwei Typen unterschieden:

- Moore-Automat
- Mealy-Automat

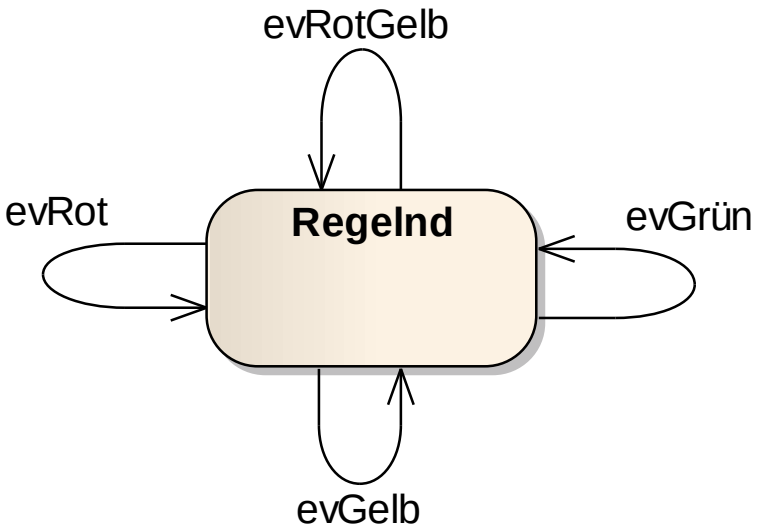
Moore-Automat (nach dem Mathematiker Edward F. Moore (1925-2003))

Ein Moore-Automat führt seine Aktionen in Abhängigkeit vom Zustand aus. Die Aktion wird ausgeführt, wenn der Zustand erreicht wird.



Mealy-Automat (nach George H. Mealy)

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



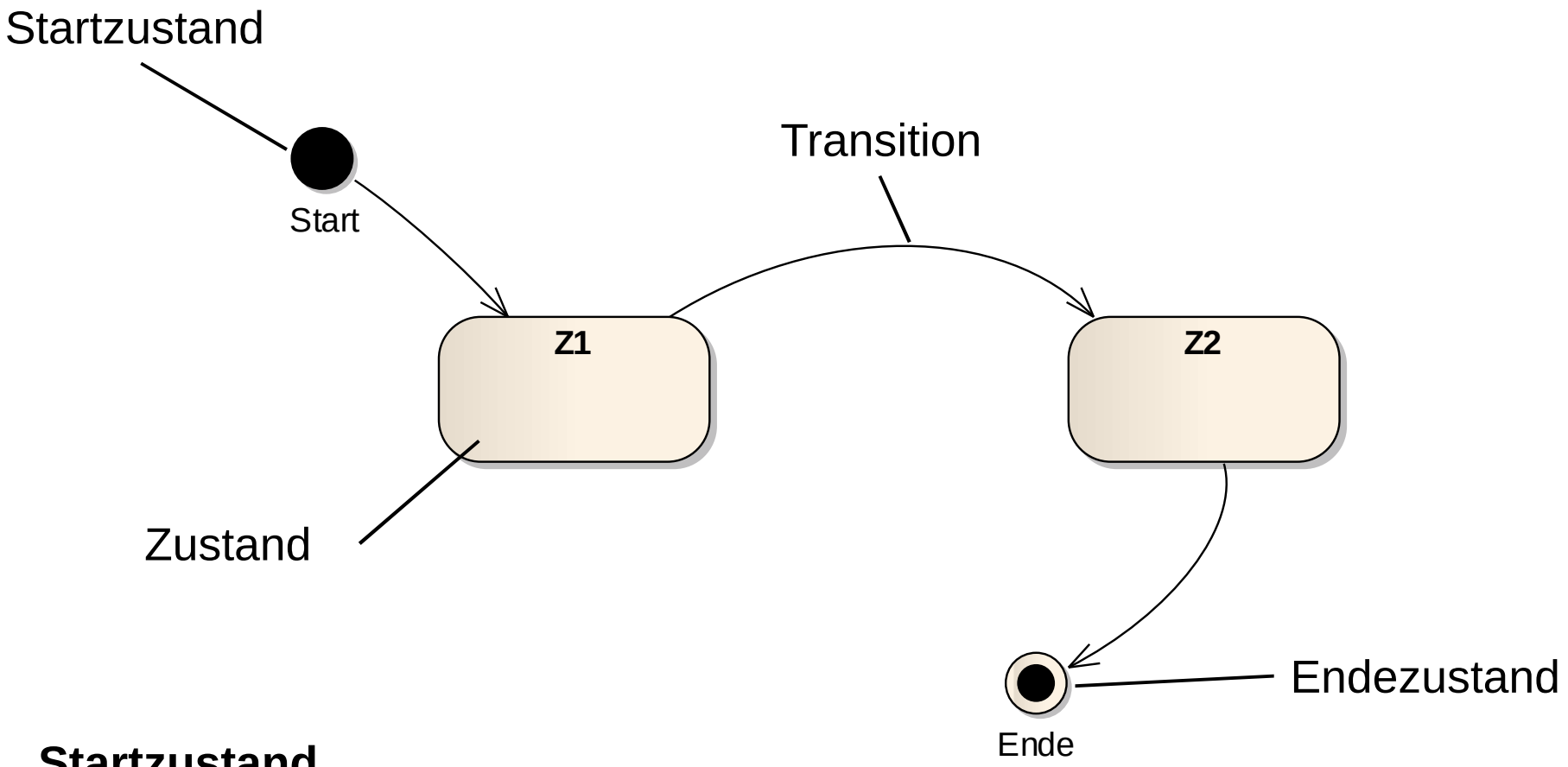
Moore oder Mealy?

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



Startzustand

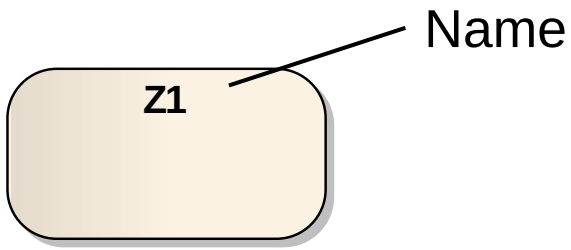
Gibt an, mit welchem Zustand der Automat startet (hier Z1).
Jeder Automat hat genau einen Startzustand.

Endezustand

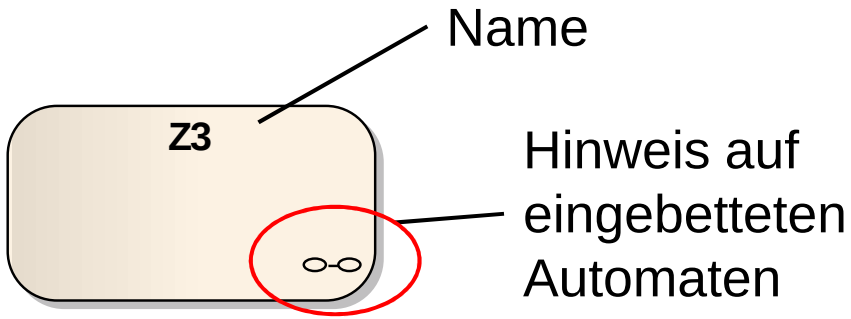
Kennzeichnet das Ende des Automaten.
Ein Automat kann beliebig viele Endezustände haben (auch keinen).

Zustand

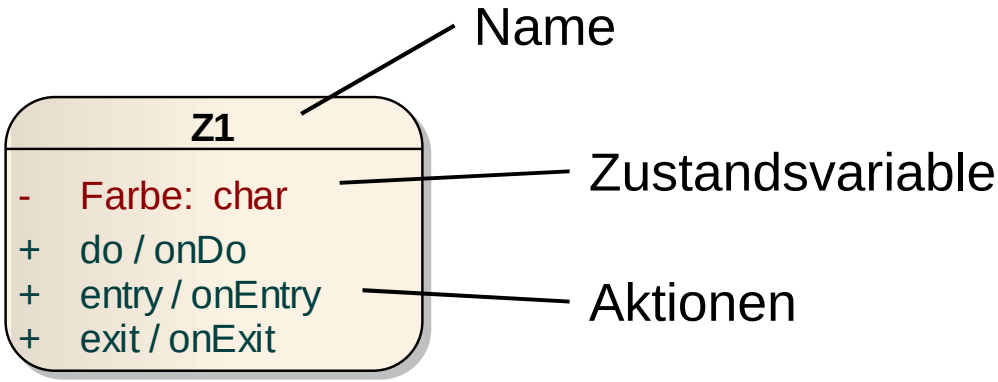
Einfache Darstellung

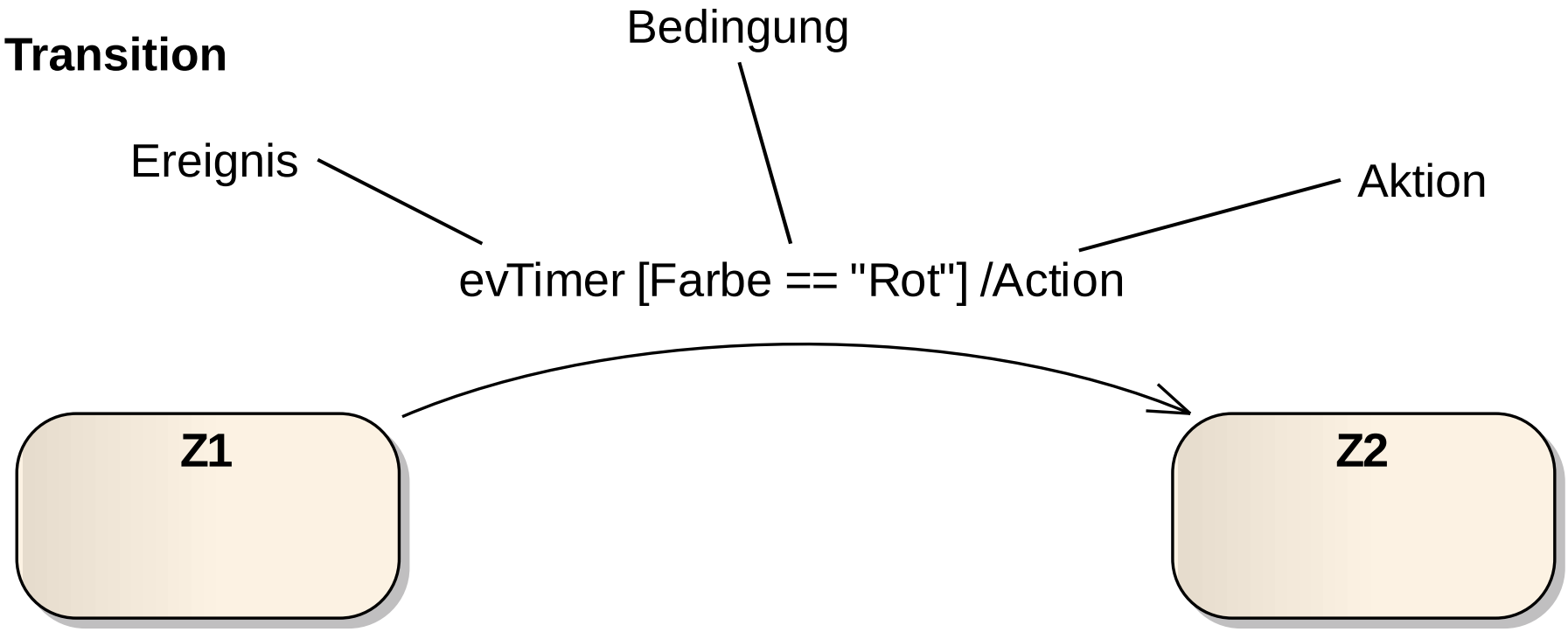


Zusammengesetzter Automat



Ausführliche Darstellung





Ereignis – Triggert den Zustandswechsel.

Bedingung – Der Wechsel erfolgt nur, wenn die Bedingung erfüllt ist.

Aktion – Bei diesem Wechsel auszuführende Aktion (Mealy-Automat)

Keine der Angaben ist Pflicht, sie können beliebig kombiniert werden.

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

- Dafür reicht ein einfaches Zeichentool.
- Auch 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.

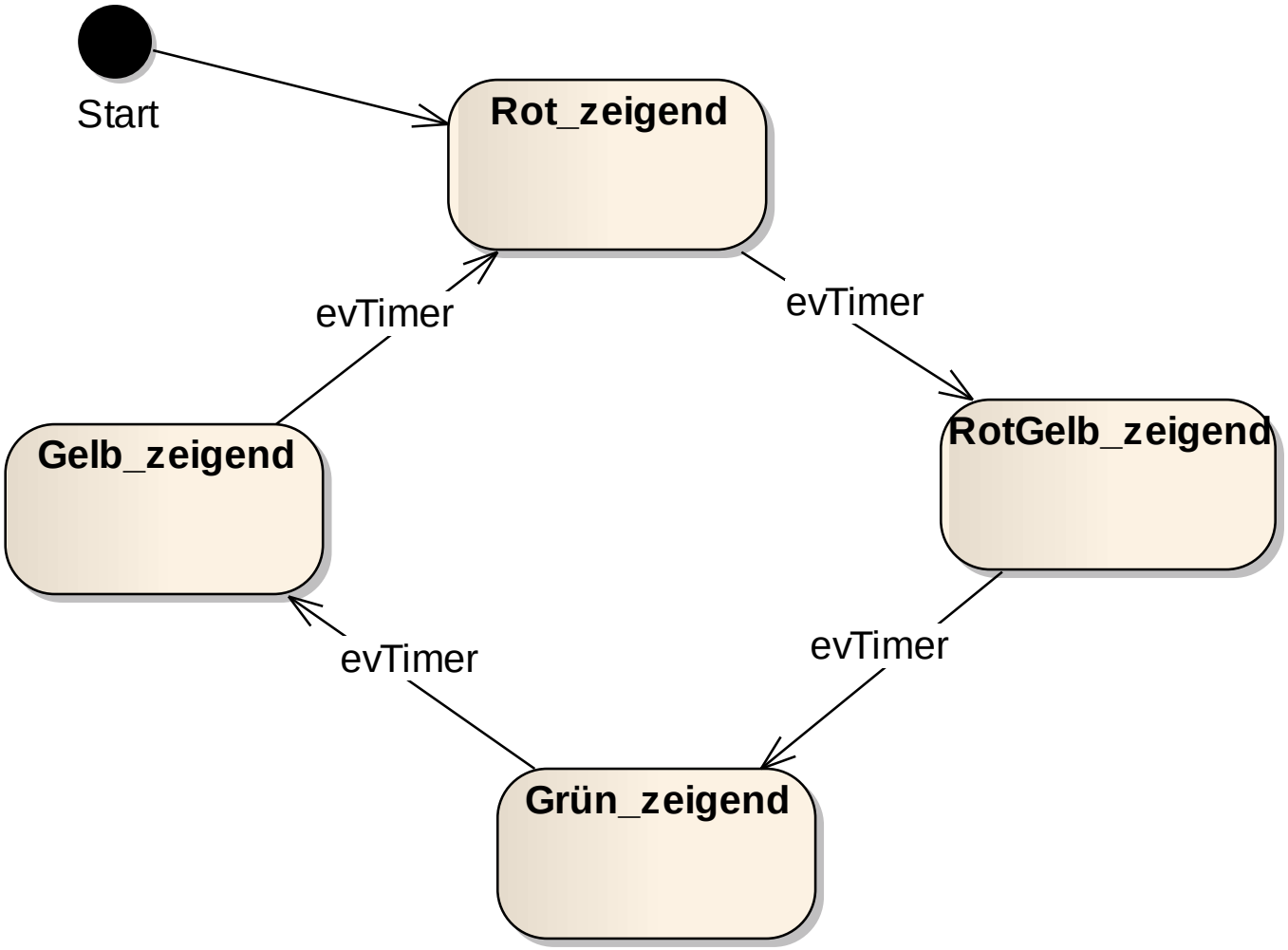
Benennung der Zustände:

- ein Zustand ist selten statisch, meistens werden Aktionen ausgeführt, wenn sich das System in diesem Zustand befindet. Das sollte sich auch im Namen des Zustands widerspiegeln.

  → **Benutze Verben in der Verlaufsform**.

- Wichtig ist, daß das Diagramm ohne Mißverständnisse gelesen werden kann.

Zustandsfolgediagramm für die Ampelsteuerung



Zustandstabelle für die Ampelsteuerung

Next State / State		Rot_zei...	RotGelb...	Grün_ze...	Gelb_z...	Start
		S0	S1	S2	S3	S4
Rot_zei...	S0		evTimer			
RotGelb...	S1			evTimer		
Grün_ze...	S2				evTimer	
Gelb_z...	S3	evTimer				
Start	S4	———				

Neue Aufgabe:

Steuern Sie die Lampen einer Ampel an.

Die Umschaltung erfolgt immer nachdem ein Zeitimpuls eingetroffen ist.

Es gibt zwei Betriebsarten – Blinkbetrieb und Regelbetrieb.

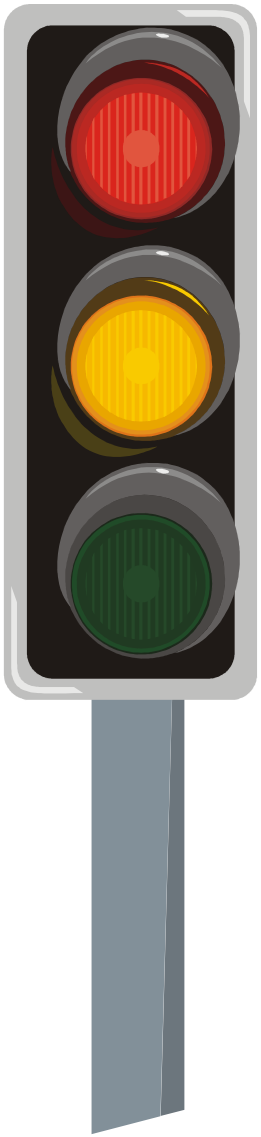
Ampelphasen Blinkbetrieb:

- Aus
- Gelb

Ampelphasen Regelbetrieb:

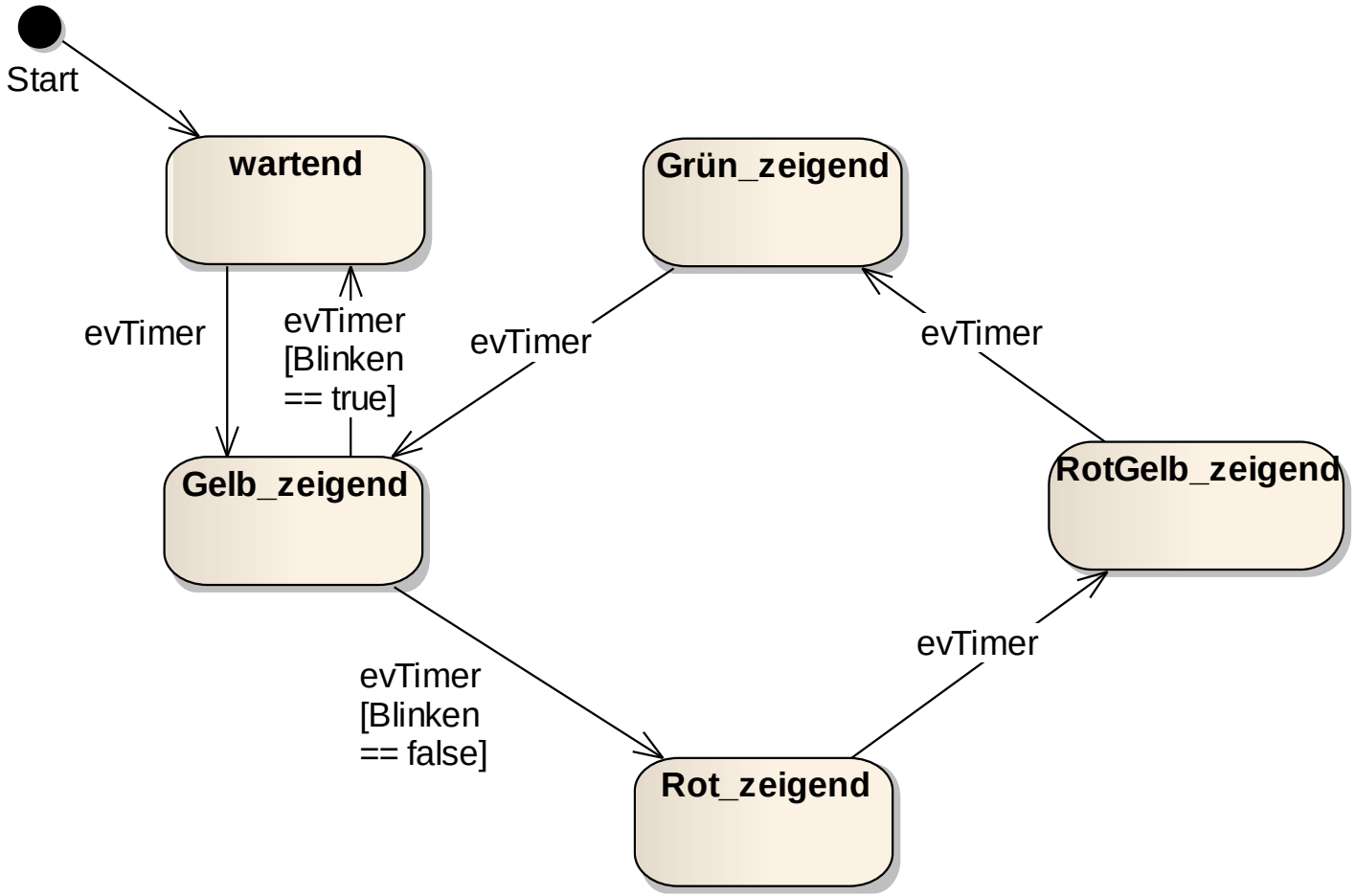
- Rot
- Rot-Gelb
- Grün
- Gelb

Nach dem Einschalten soll sich die Ampel im Blinkbetrieb befinden.



Erweiterungen:

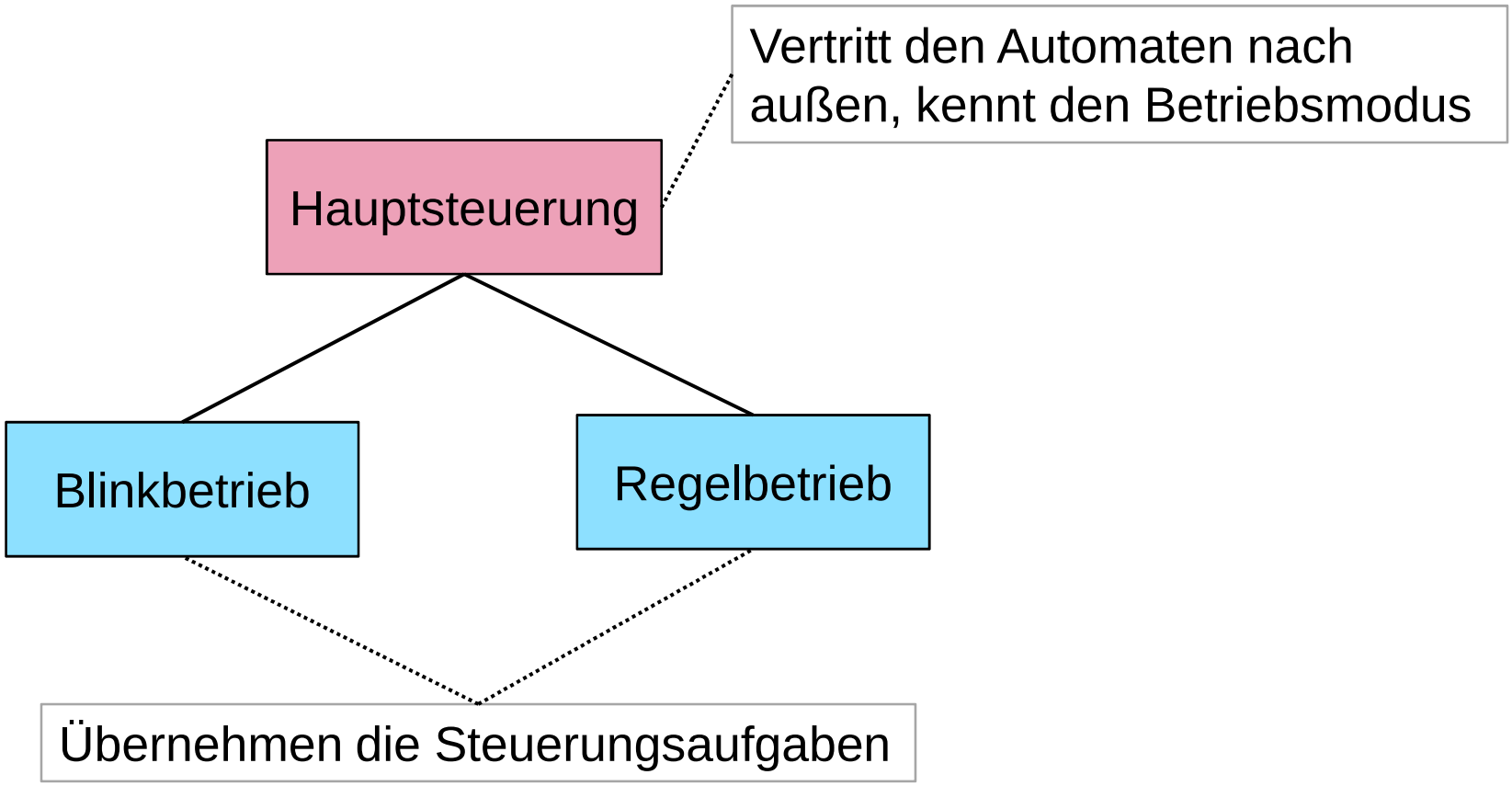
- Flag "Blinken" zum Unterscheiden zwischen Regel- und Blinkbetrieb.
- Ein zusätzlicher Zustand.



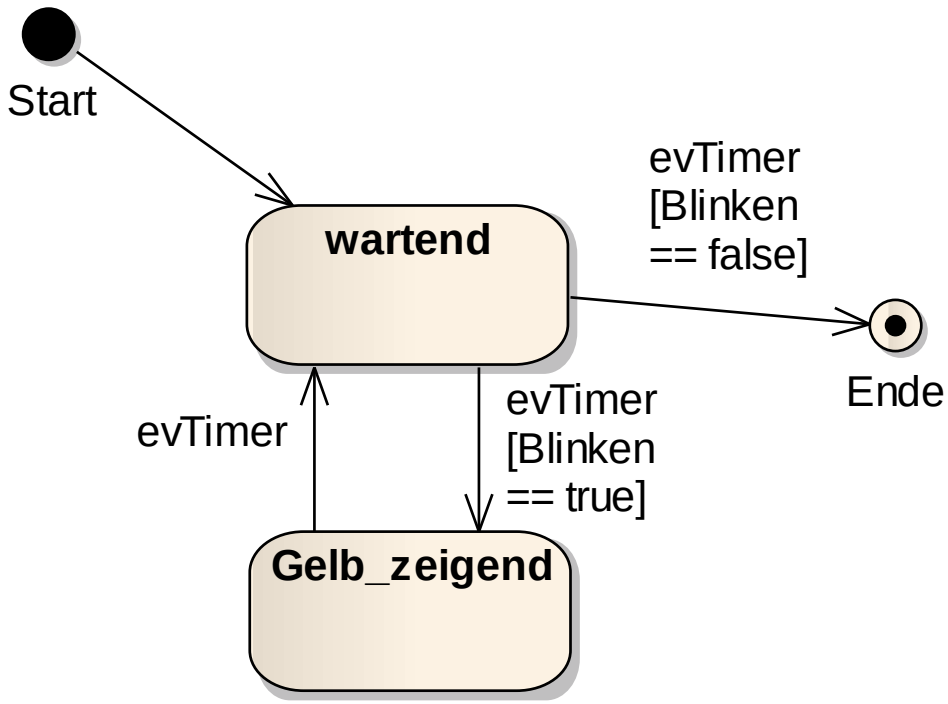
Probleme:

- Der Automat wird unübersichtlich.
- Es ist nicht genau festzustellen, in welchem Betriebsmodus der Automat gerade arbeitet (der Wert des Flags "Blinken" ist nicht ausreichend).
- Weitere Erweiterungen werden immer schwieriger.
- Fehlersuche wird komplizierter.

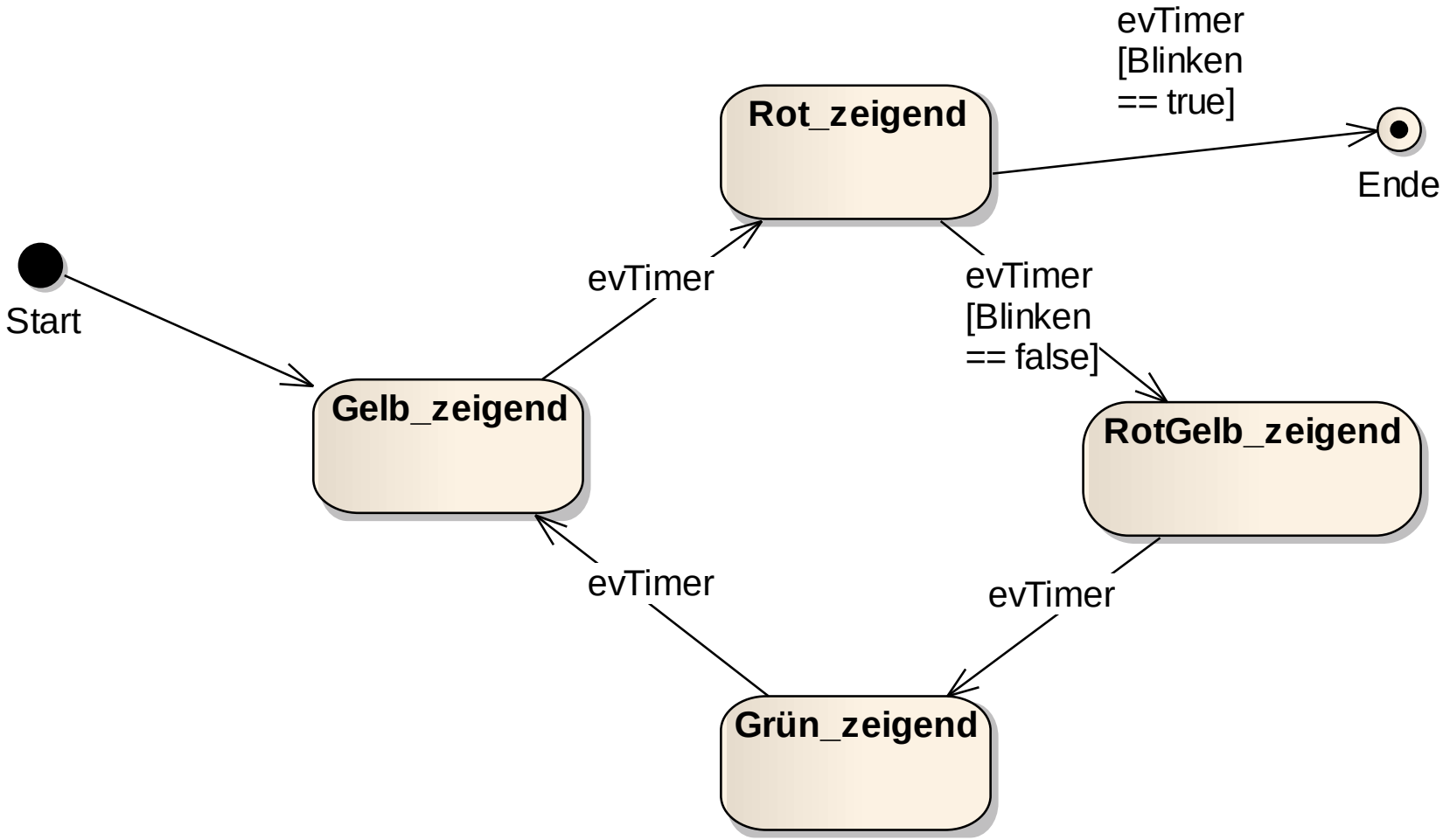
Um Klarheit zu schaffen, muß eine Hierarchie von Automaten gebildet werden.



Der Automat für den Blinkbetrieb

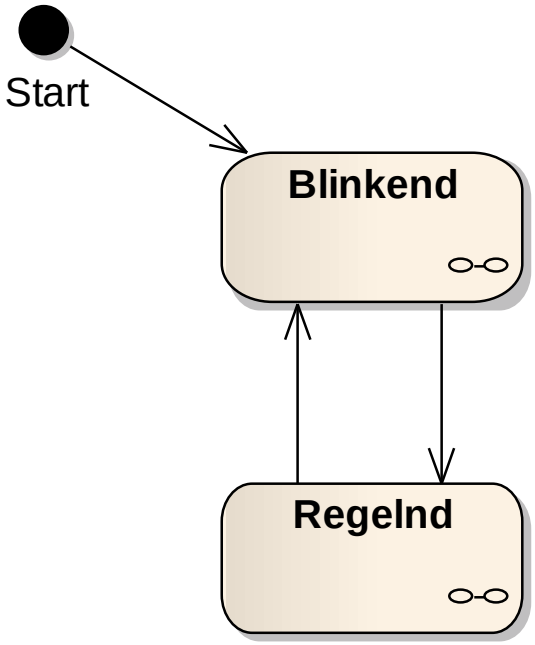


Der Automat für den Regelbetrieb



Der "Chef"-Automat

Der Betriebsmodus ist nun klar erkennbar.



An den Transitionen sind keine Bedingungen oder Ereignisse nötig, der Zustandswechsel wird immer durchgeführt, wenn ein untergeordneter Automat beendet wurde.

Welches Modell ist komplizierter?

Flacher Automat:

- 5 Zustände
- 25 mögliche Transitionen
- Funktion ist nicht klar erkennbar

Hierarchischer Automat:

- 8 Zustände insgesamt
- 24 mögliche Transitionen
- Klare Aufgabenverteilung
- Einfach erweiterbar

Wieviel Zustände darf ein Automat haben?

Da die Anzahl der Transitionen quadratrisch wächst, sollte die Anzahl der Zustände nicht zu hoch sein.

Richtwert ist die „Magische Sieben“. Grundlage ist die Erkenntnis, daß ein durchschnittlicher Mensch sieben Dinge gleichzeitig erfassen kann.

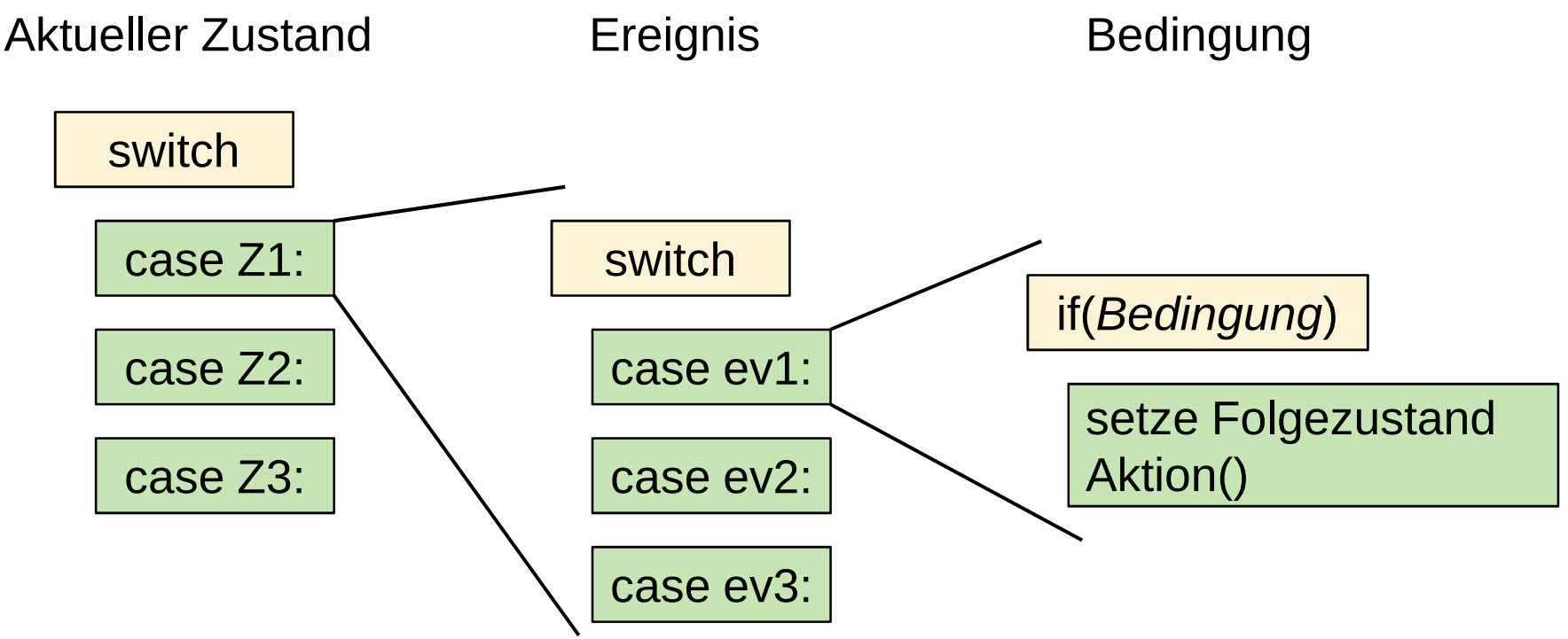
Allerdings ist die theoretische Anzahl von 49 Transitionen schon recht viel (auch eine nicht vorhandene Transition kann Probleme bereiten).

Deshalb immer prüfen, ob der Automat zerlegt werden kann. – Eine hohe Anzahl an Zuständen ist meistens ein Zeichen dafür, daß mehrere Automaten miteinander vermischt wurden.

Programmierung von endlichen Automaten

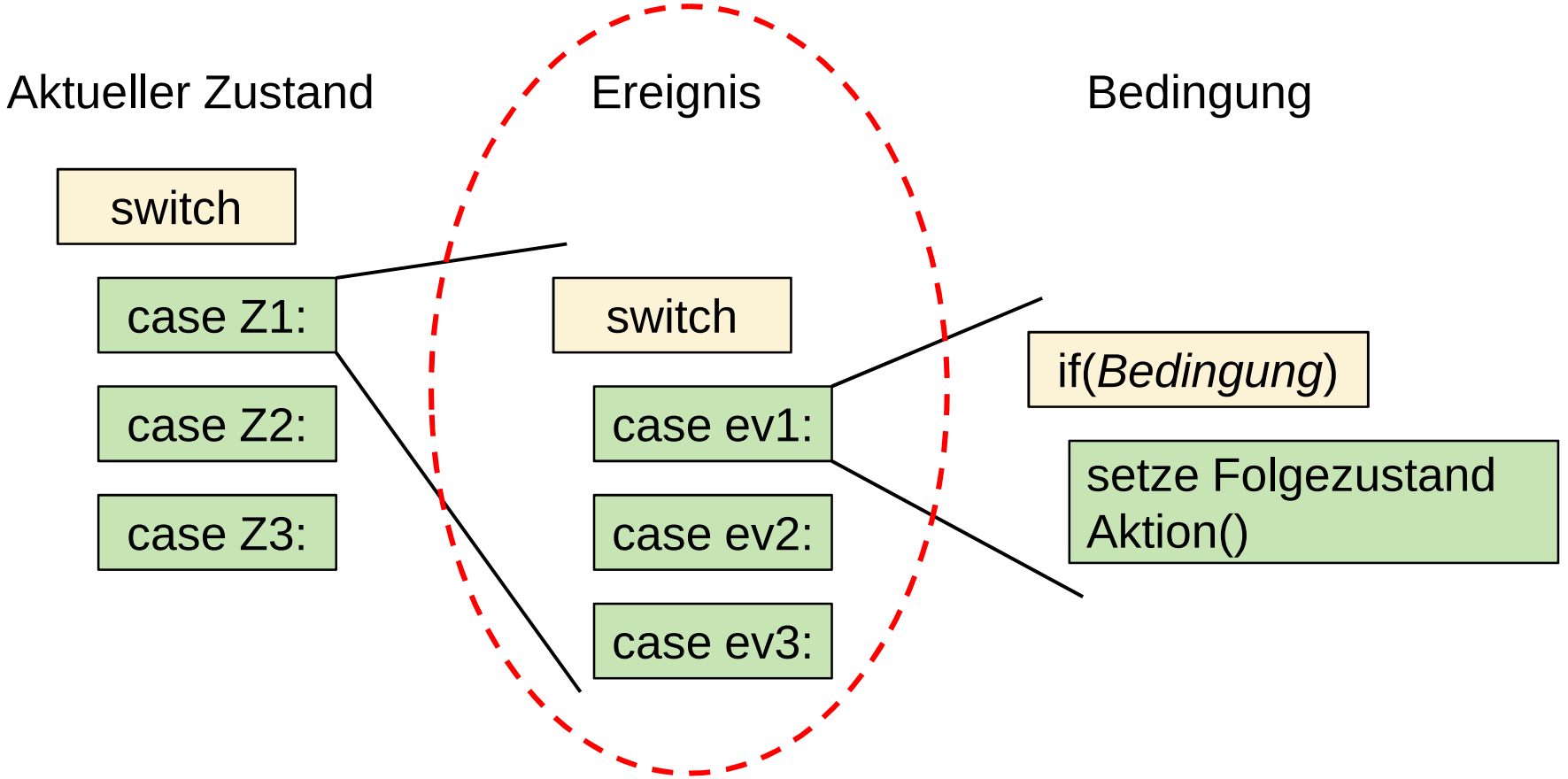
MICRO CONSULT

Klassische switch-case-Implementierung



Da in unserem Beispiel nur ein Ereignis (der Zeittakt) existiert, vereinfacht sich die Implementierung.

→ Die Auswertung der Ereignisse kann weggelassen werden.



Vorteile

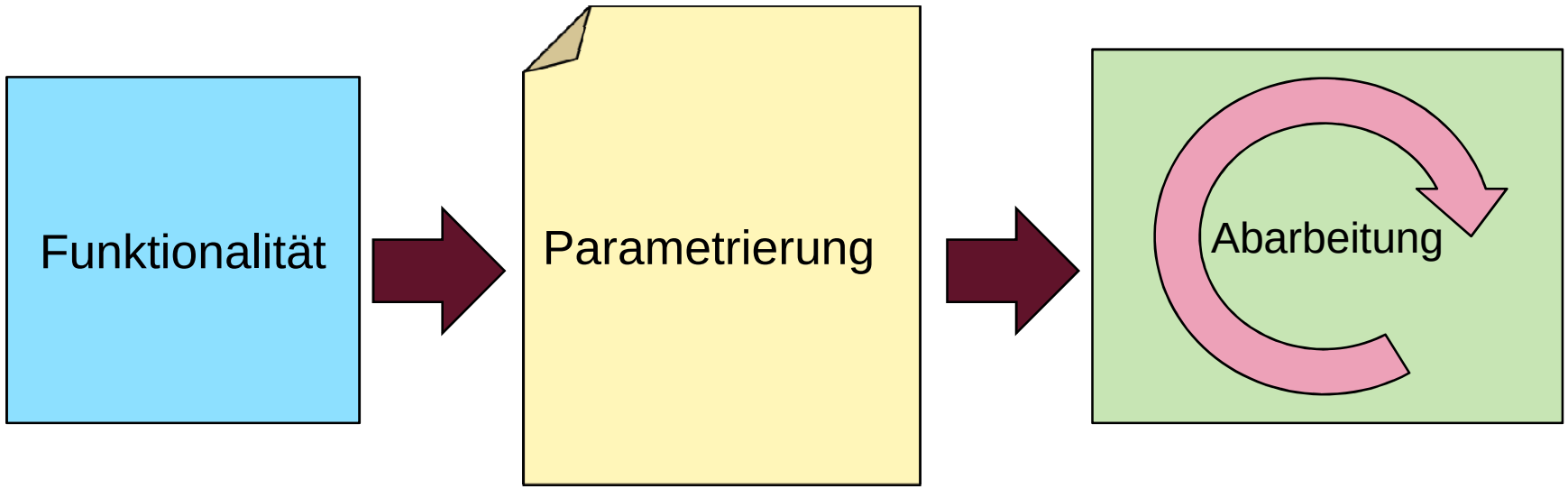
- einfaches Prinzip

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

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.



Stellvertretend für viele Implementierungen werden hier zwei Möglichkeiten der Programmierung von Automaten gezeigt.

- Einfacher tabellengesteuerter Automat
- Automat nach dem Zustands-Entwurfsmuster (State Pattern)

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

Aufteilung:

Beschreibung des Automaten

→ Zustand-Ereignis-Tabelle

- Enthält die Beschreibungen für alle Transitionen des Automaten.

Mechanik des Automaten

→ Funktion zur Abarbeitung der Tabelle

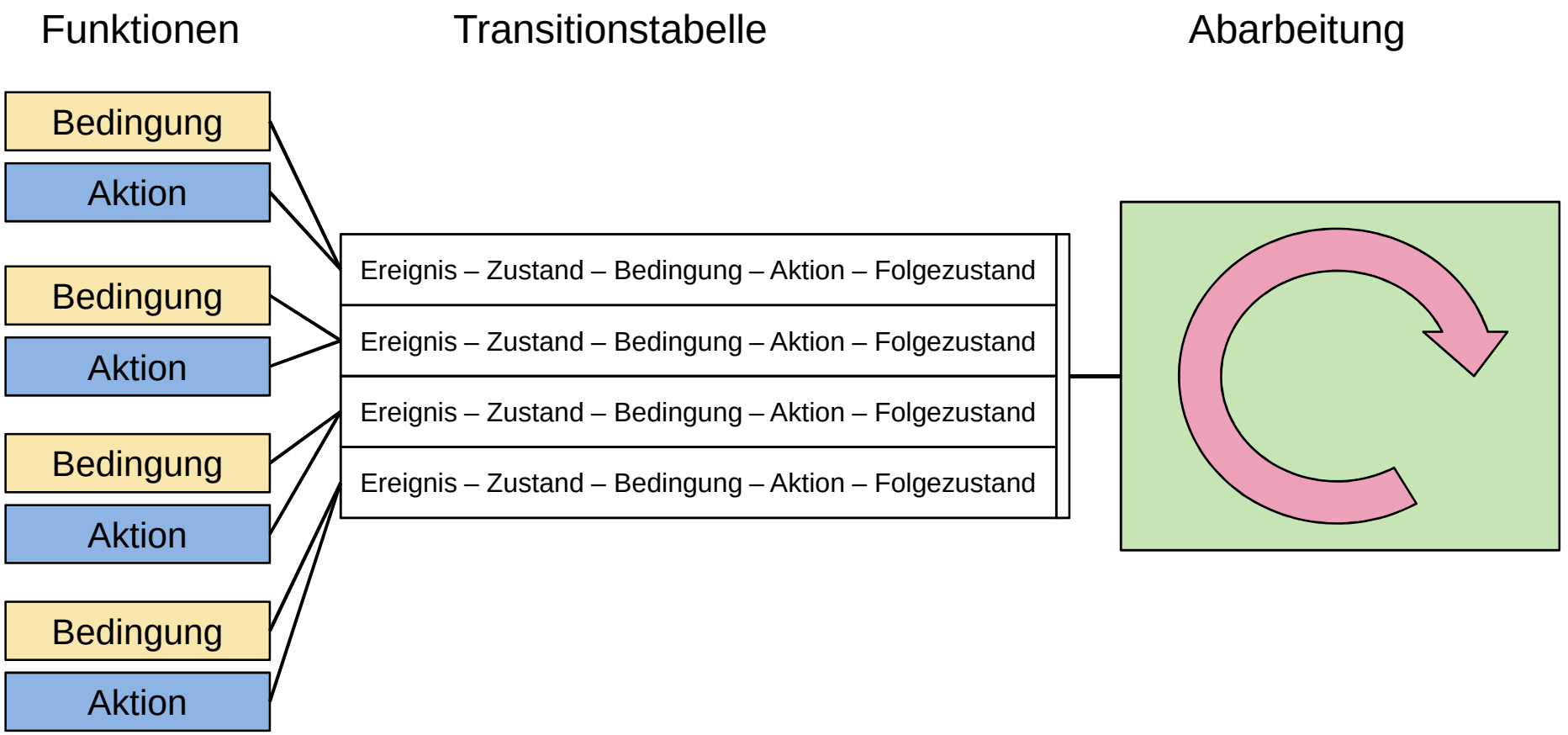
- Durchsucht die Tabelle nach einem passenden Eintrag.
- Arbeitet alle Anweisungen des Eintrags ab.

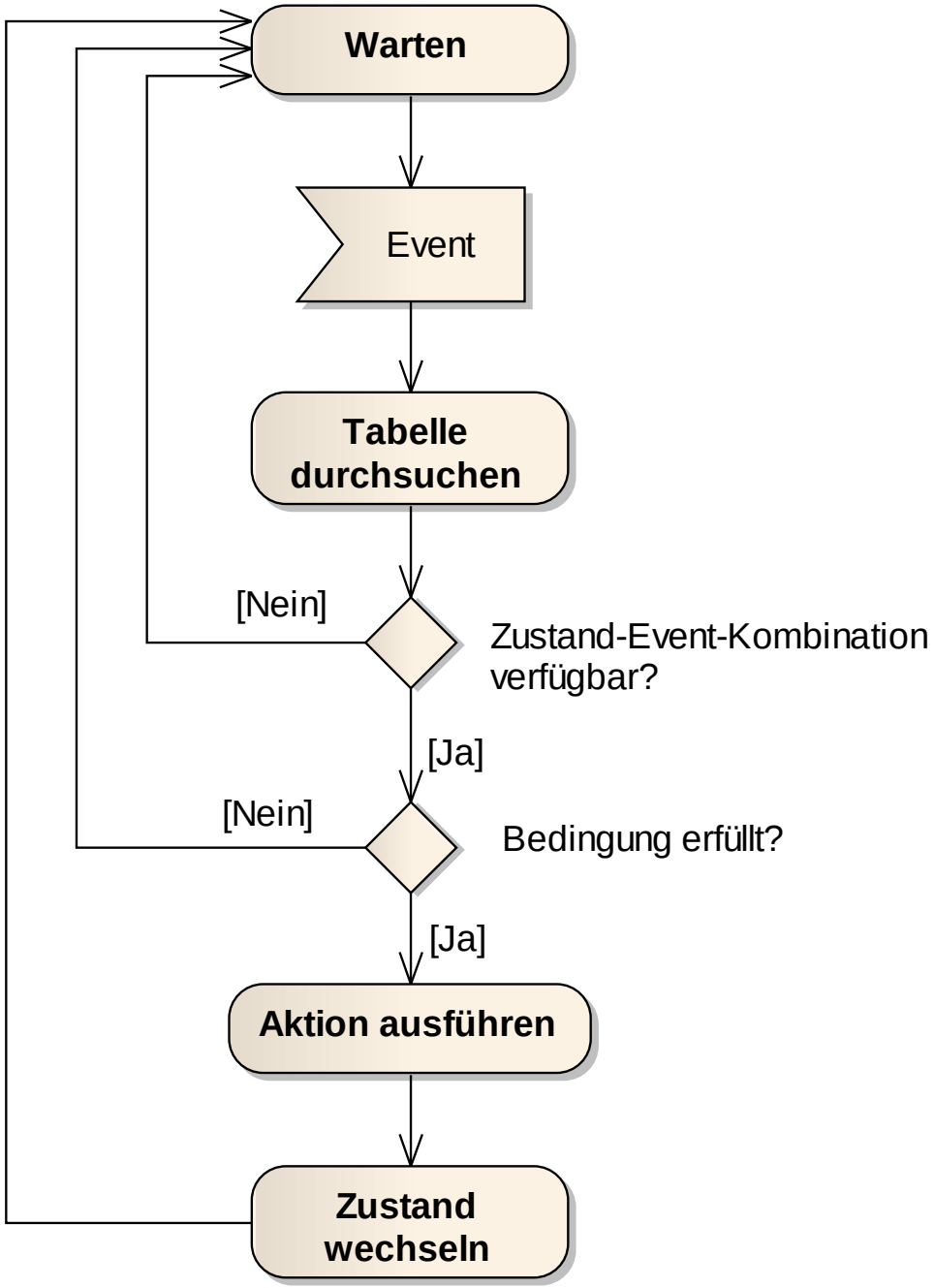
Funktionalität des Automaten

→Funktionen des eigentlichen Programmes

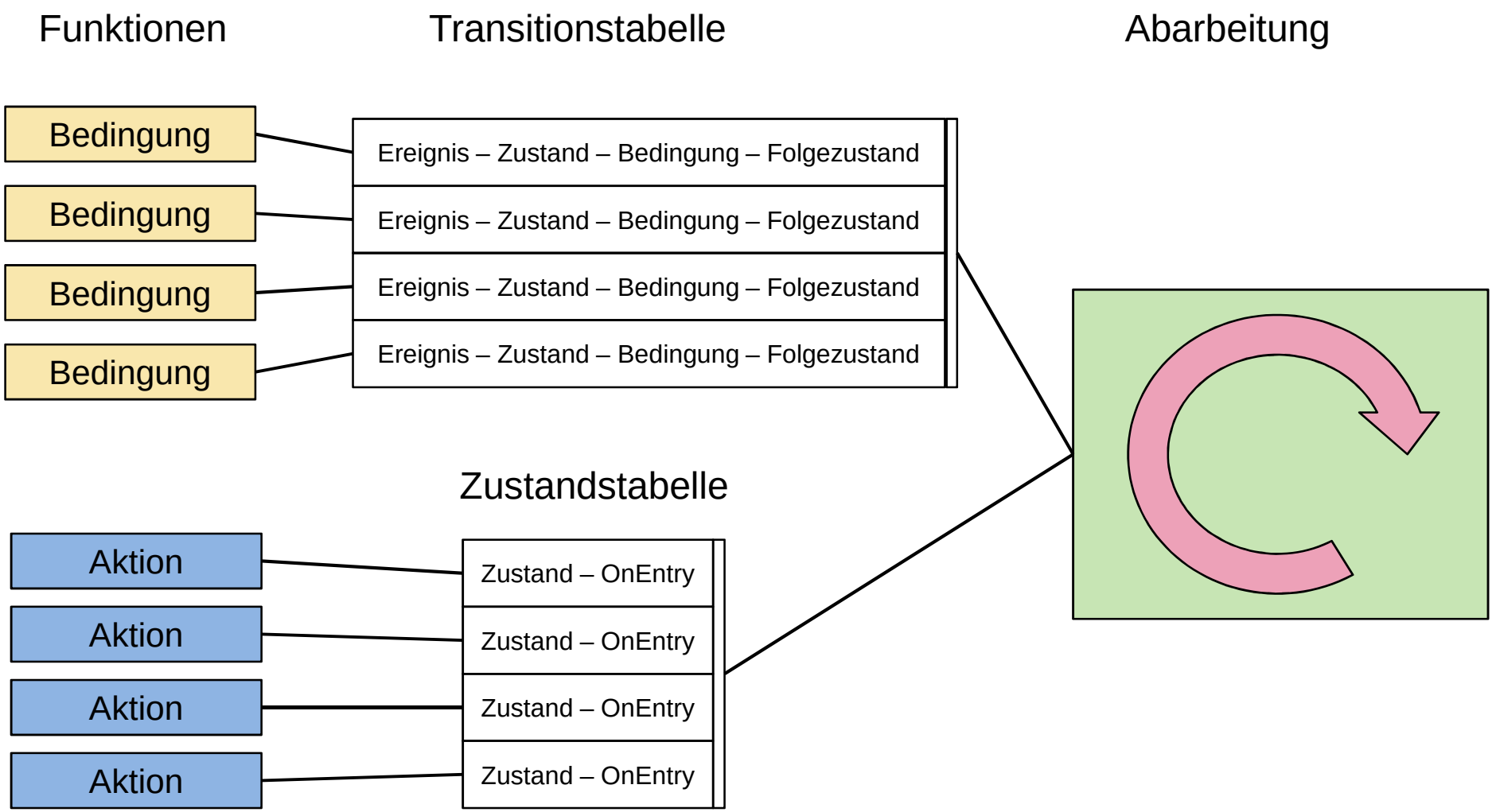
- Zu prüfende Bedingungen
- Auszuführende Aktionen

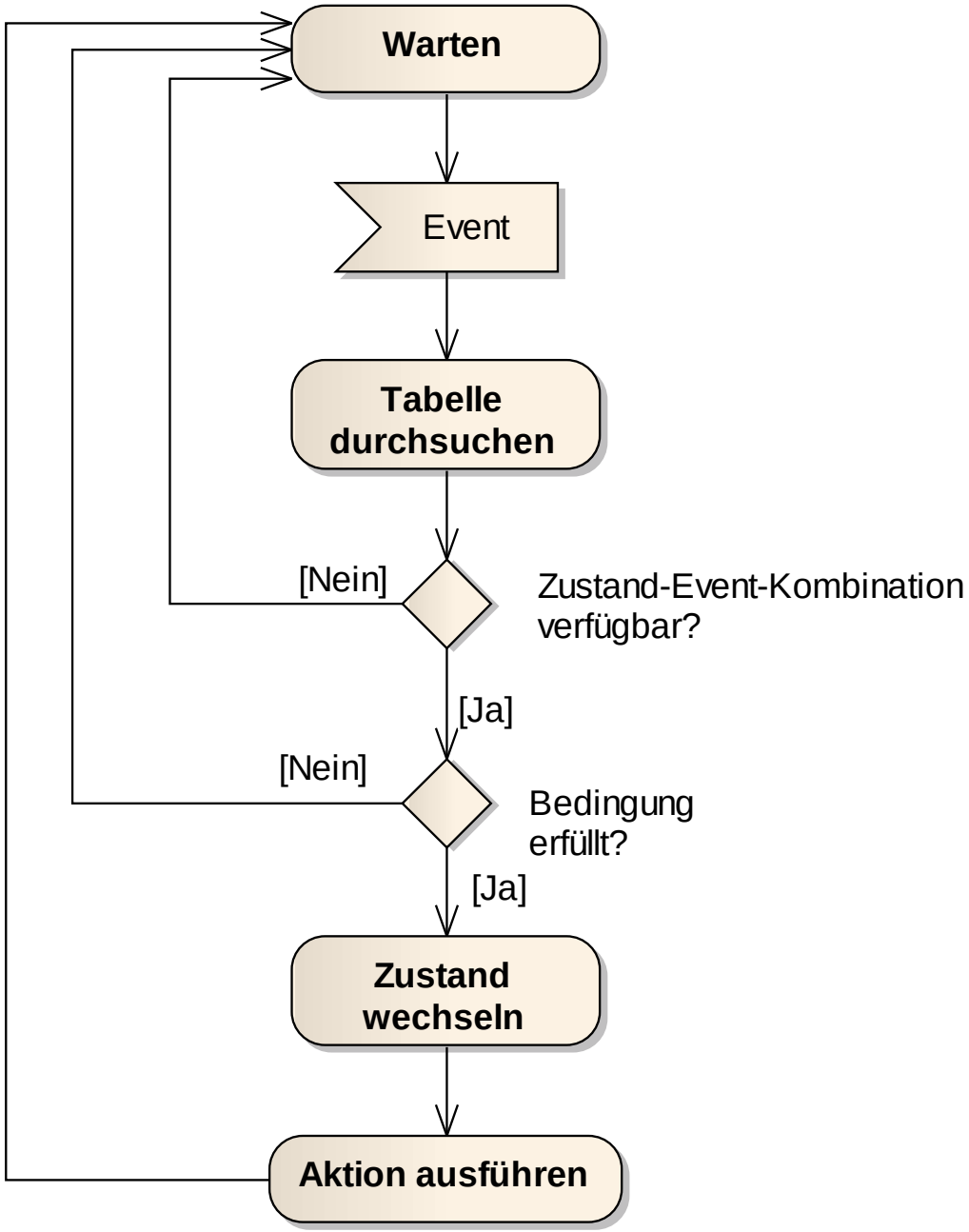
Mealy-Automat




Warten
Event
Tabelle durchsuchen
[Nein]
Zustand-Event-Kombination verfügbar?
[Ja]
[Nein]
Bedingung erfüllt?
[Ja]
Aktion ausführen
Zustand wechseln

Moore-Automat




Warten
Event
Tabelle durchsuchen
[Nein]
Zustand-Event-Kombination verfügbar?
[Ja]
[Nein]
Bedingung erfüllt?
[Ja]
Zustand wechseln
Aktion ausführen

Vorteile:

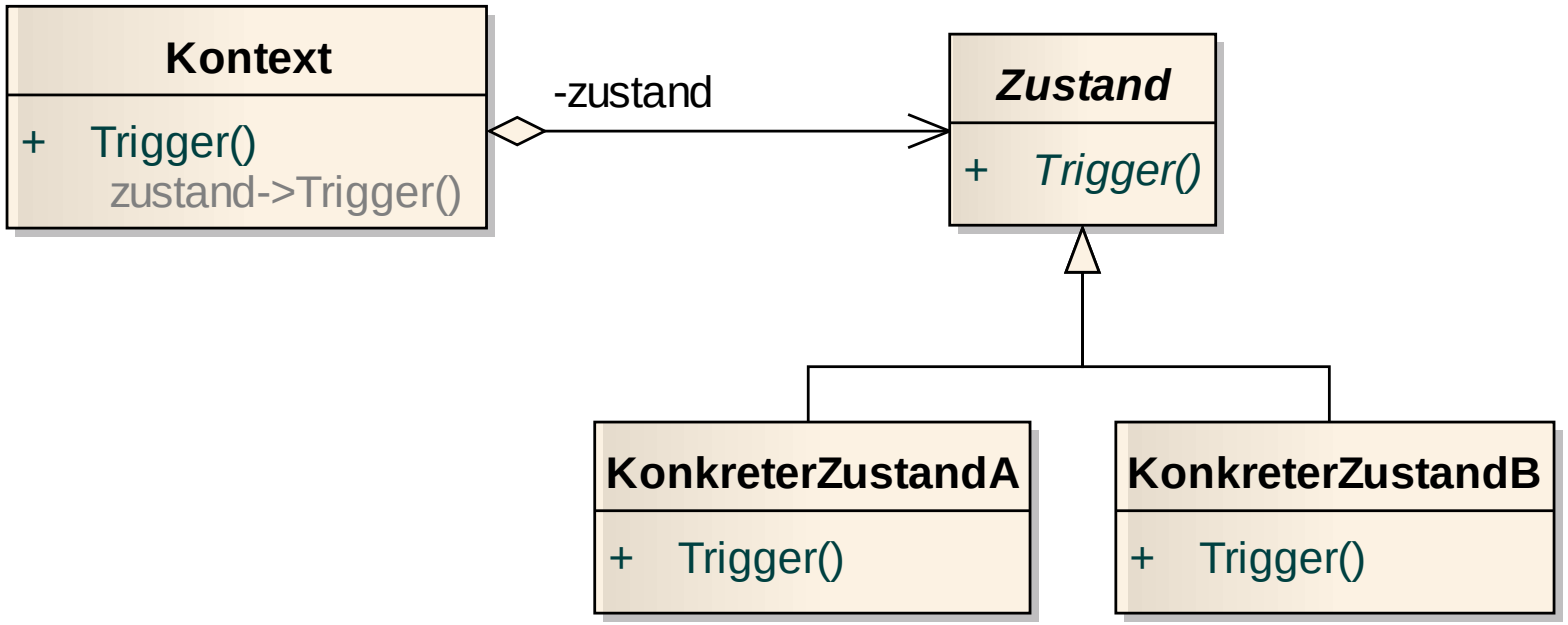
- klare Trennung von Automat und Verhalten
- einfaches Prinzip
- direkter Bezug von Transition im Modell des Automaten zu dem entsprechenden Tabelleneintrag

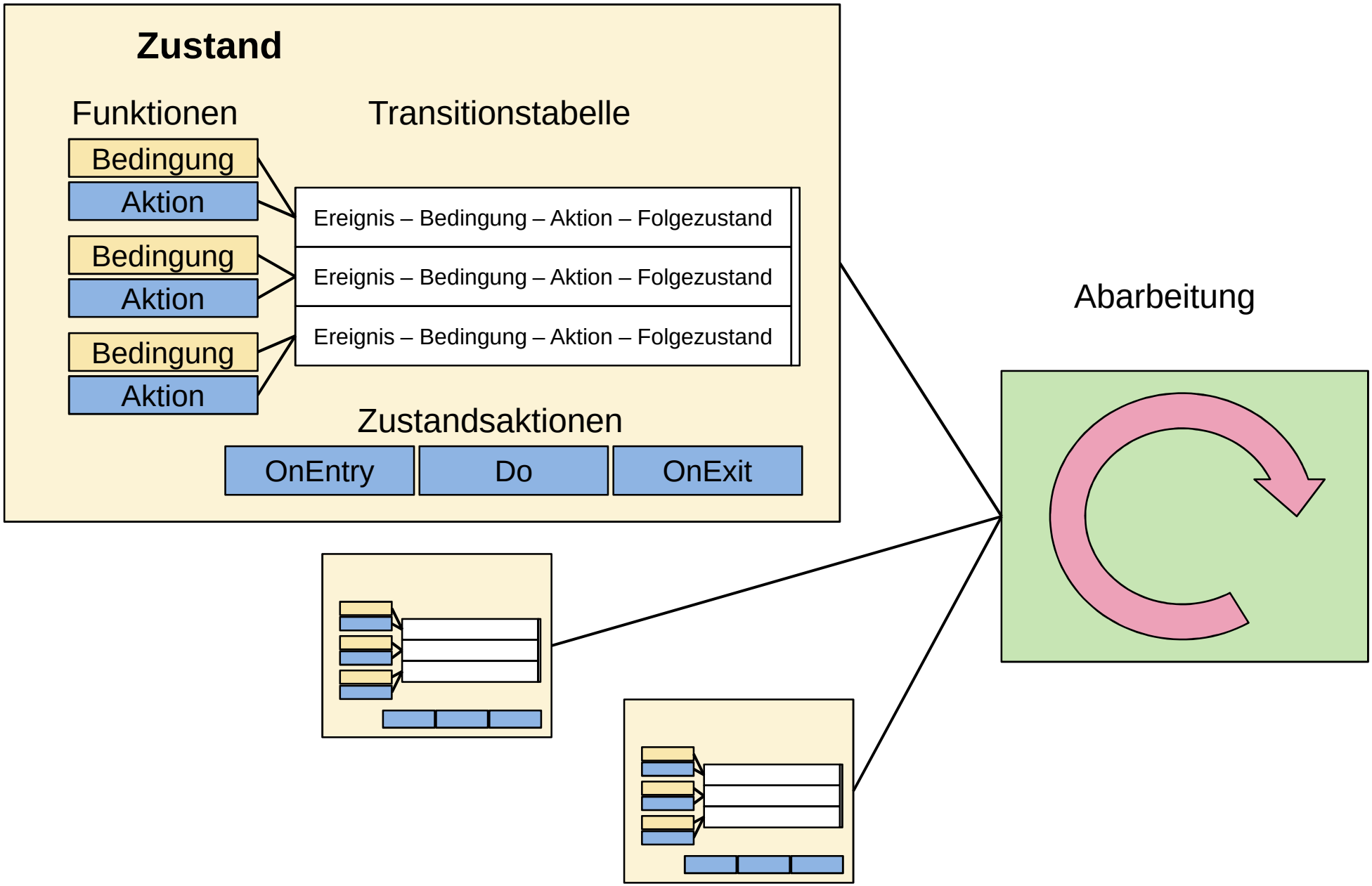
Nachteile:

- im schlimmsten Fall müssen alle Tabelleneinträge durchsucht werden

Die Implementierung ist angelehnt an das GoF State Pattern.

Der Zustand wird nicht mehr über einen Zahlenwert ausgedrückt, sondern über ein Objekt.




Zustand
Funktionen
Transitionstabelle
Bedingung
Aktion
Bedingung
Aktion
Bedingung
Aktion
Ereignis – Bedingung – Aktion – Folgezustand
Ereignis – Bedingung – Aktion – Folgezustand
Ereignis – Bedingung – Aktion – Folgezustand
Zustandsaktionen
OnEntry
Do
OnExit
Abarbeitung

Vorteile:

- klare Trennung von Automat und Verhalten
- Transitionstabelle wird in kleine Stücke unterteilt
- direkter Bezug von Transition im Modell des Automaten zu dem entsprechenden Tabelleneintrag
- Die in der UML definierten Aktionen im Zustand (OnEntry, Do, OnExit) können problemlos im Code abgebildet werden

Nachteile:

- komplexere Automatenlogik – muß aber nur einmal erstellt werden

Einsatz von Tools

Drei Gruppen von Tools können unterschieden werden:

- Zeichentools
- Compiler mit Automatengenerator
- UML-Tool mit Codegenerator

Die Entscheidung, welche Variante benutzt wird, hängt stark von den Randbedingungen im Projekt ab:

- Wie hoch ist das Budget?
- Besteht die Möglichkeit der Wiederverwendung in anderen Projekten?
- Wird das Tool auch für andere Aufgaben eingesetzt?

Unbedingt notwendig zur graphischen Entwicklung der Automaten.

Die Spanne der möglichen Tools reicht von Powerpoint (oder OpenOffice Impress) über Visio bis hin zu einfachen UML-Tools.

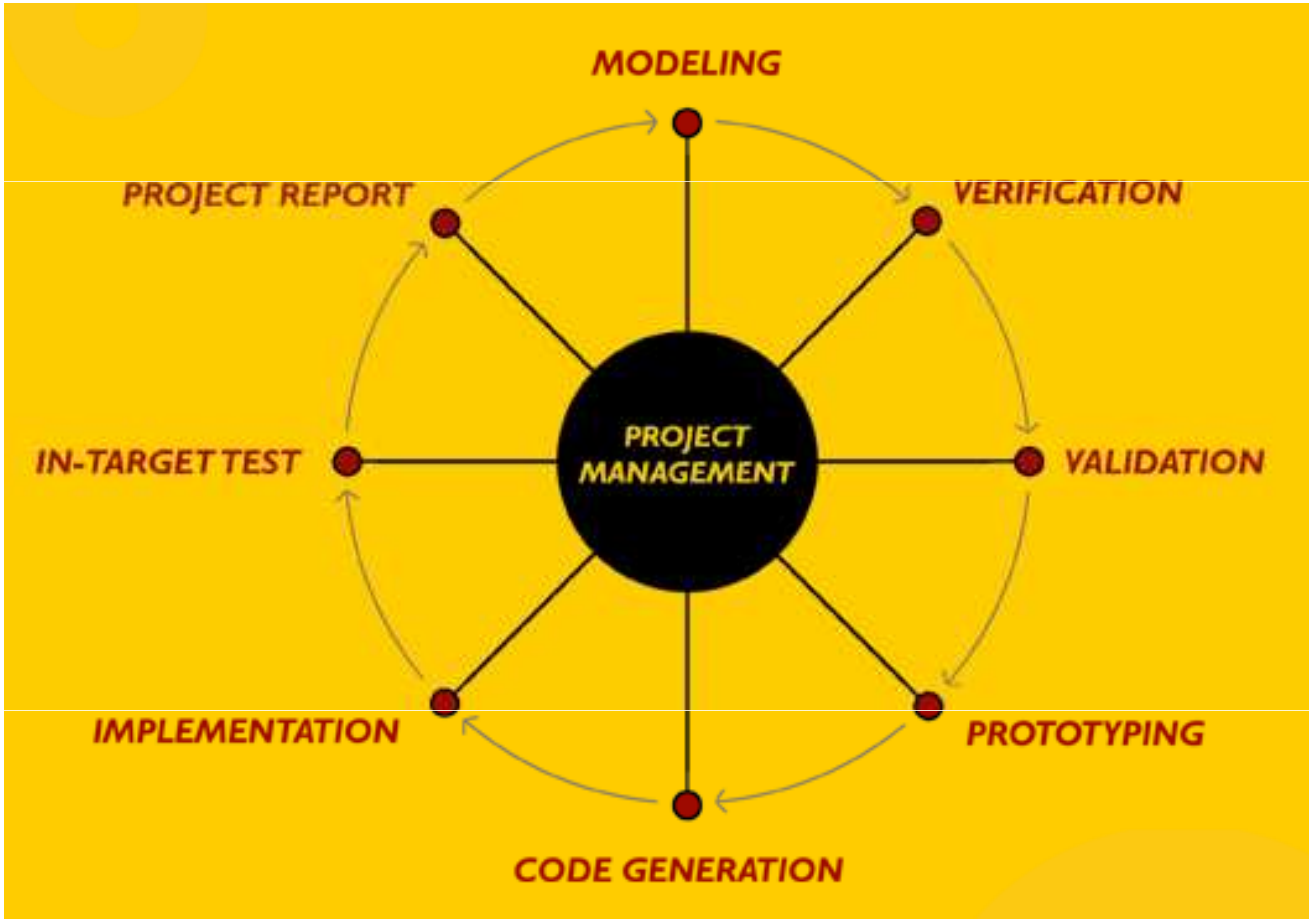
Vorteile

- preiswert
- einfach zu bedienen

Nachteile

- Leistung der Tools beschränkt sich auf die graphische Darstellung

Modellierung von Automaten in spezialisierten Werkzeugen.



Die Unterstützung der Werkzeuge geht weit über den graphischen Entwurf hinaus.

Die kompakte Lösung für die Verwendung von Automaten im Projekt.

Vorteile

- Leistungsfähiger graphische Editor
- Prüfung der Automatensyntax
- Simulation der Automaten
- Generierung des Quellcodes
- Visuelles Debuggen (Animierter Automat zeigt den Zustand des Targets)
- Codegenerierung paßt auf jedem Fall zum Target

Nachteile

- Beschränkung auf Modellierung von Automaten

Die "Alles in Einem"-Lösung

Vorteile

- Von der Anforderung bis zum Code – alles in einem Werkzeug
- Prüfung der Automatensyntax
- Simulation der Automaten
- Generierung des Quellcodes
- Visuelles Debuggen (Animierter Automat zeigt den Zustand des Targets)

Nachteile

- Hohe Anschaffungskosten
- nicht zu unterschätzende Einarbeitungszeit (UML-Syntax + Bedienung)
- Nicht für alle Targets verfügbar bzw. Anpassung kostet extra

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

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

Auch mit der Programmiersprache C können moderne objektorientierte Automaten programmiert werden.

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

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

http://download.microconsult.net/ESE2009/Automaten.zip

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

Buchtip:

"Practical UML Statecharts in C/C++"

Miro Samek

978-0-7506-8706-5