

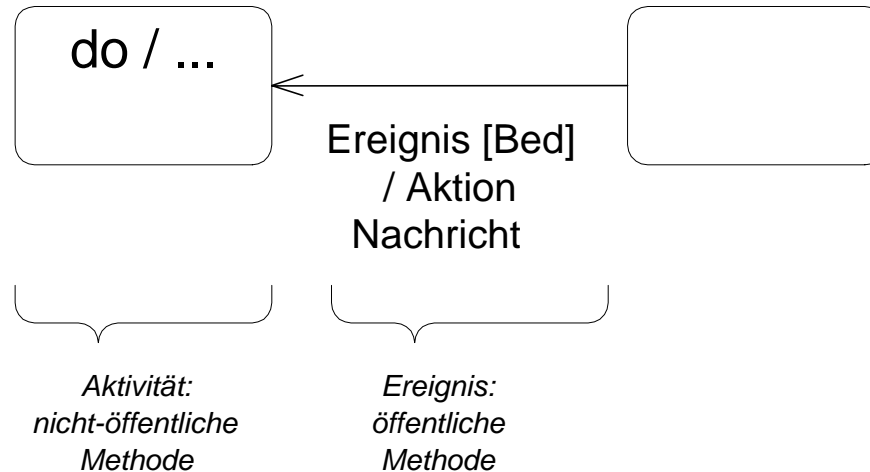


7. Codegenerierung Zustandsdiagramme

von
Prof. Dr. Wolfgang Weber

Umsetzung von Zustandsdiagrammen

- Abbildung auf Attribute, Operationen und Methoden der zugrundeliegenden Klasse



Wann Codierung der Aktionen und Aktivitäten ?

- schon im OOD
- erst nach der Umsetzung (ausfüllen der erzeugten Coderahmen)
- beides



Umsetzung von Zustandsdiagrammen

Sinnvoll:

Auch Rückwärtsabbildung von Code \Rightarrow OOD

= „**Round-Trip-Engineering**“

Frage: Muss sich ein Objekt seinen Zustand merken?

- Da Zustandsübergänge abhängig vom Zustand sind, muss das Objekt wissen, in welchem aktuellen Zustand es sich befindet.

=> **Abspeicherung des aktuellen Zustandes in zusätzl.
Attribut "state"**

(z.B. realisiert als Aufzählungstyp der Zustandsnamen)



Ablauf beim Eintreffen eines Ereignisses

Prüfung, in welchen Zustand sich das Objekt befindet.

1. Prüfung, ob dort eine Transition mit diesem Ereignis beginnt.

Falls ja:

- ❑ Bedingung der Transition prüfen (erfüllt?)
- ❑ Wenn erfüllt, state-Attribut auf Folgezustand setzen.
- ❑ Aktion der Transition ausführen.
- ❑ Nachrichten der Transition versenden
- ❑ Aufrufen des do-Teils des Folgezustandes.

Falls nein:

- ❑ Ignorieren des Ereignisses

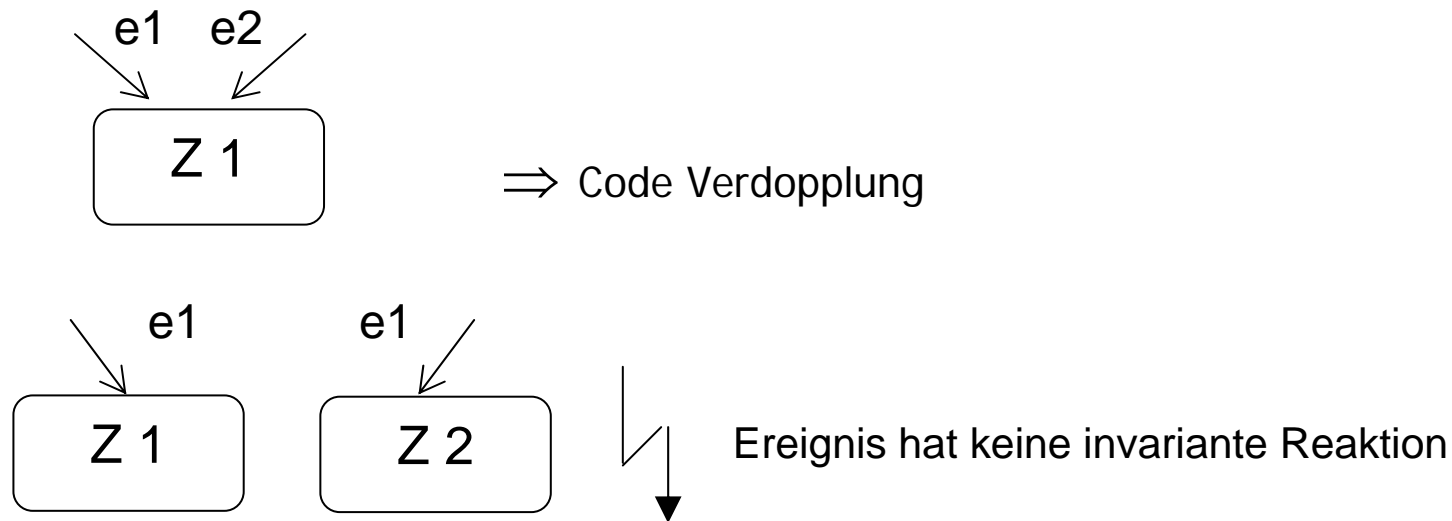
Frage:

- *Warum muss der do-Teil in eine gesonderten Methode ausgelagert sein, warum kann der do-Teil nicht direkt im Rumpf der Ereignismethode programmiert sein?*

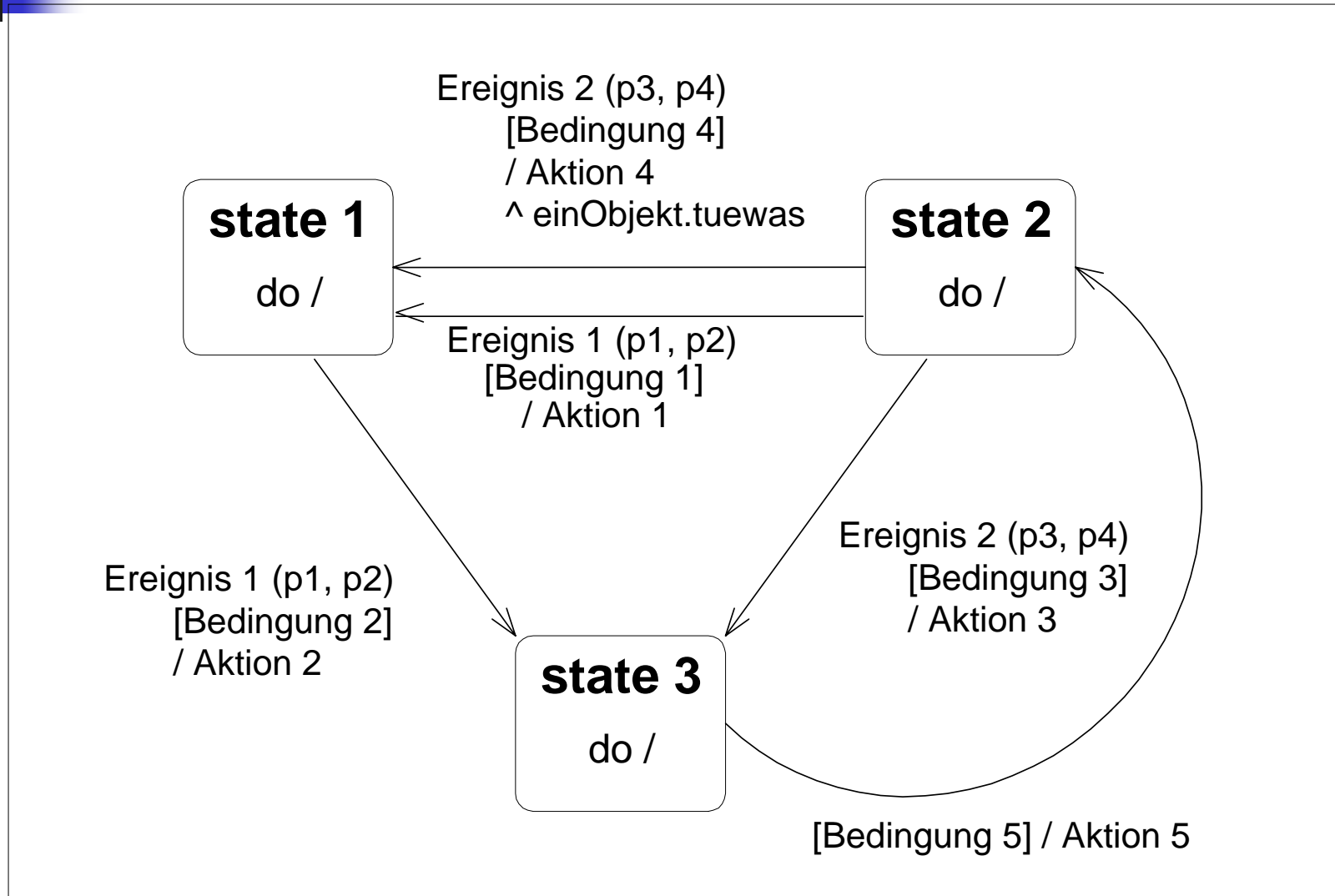
Codegenerierung Zustandsdiagramme

Frage:

- *Warum muss der do-Teil in eine gesonderten Methode ausgelagert sein, warum kann der do-Teil nicht direkt im Rumpf der Ereignismethode programmiert sein?*



Codegenerierung Zustandsdiagramme





Die Umsetzung in C++

```
virtual void Ereignis1 (anytype p1, anytype p2)
{
    switch (state)
    {
        case s1: if (<Bedingung 2>)
        {
            state = s3;
            <Aktion 2>
            Zustand_s3 (p1, p2);
        }
        break;
        case s2: if (<Bedingung 1>)
        {
            state = s1;
            <Aktion 1>
            Zustand_s1 (p1, p2);
        }
        break;
    } // switch
};
```



Die Umsetzung in C++

```
void Ereignis2 (anytype p3, anytype p4)
{
    switch (state)
    {
    case s2:    if (<Bedingung 3>)
                {
                    state = s3;
                    <Aktion 3>
                    Zustand_s3 (p3, p4);
                }
                else
                if (<Bedingung 4>)
                    {
                        state = s1;
                        <Aktion 4>
                        <Nachricht tuewas an Objekt einObjekt>
                        Zustand_s1 (p3, p4);
                    }
                break;
    } // switch
};
```



Die Umsetzung in C++

Frage:

- Was geschieht in Funktionen
 - Zustand_s1 (p_i, p_j); und
 - Zustand_s3 (p_k, p_n);



Die Umsetzung in C++

```
void Zustand_s1 (anytype pi, anytype pj)
{
    <do-Teil Zustand s1>      // durch Aktivitätendiagramm
                              // definiert und eingebettet
```



Die Umsetzung in C++

```
void Zustand_s3 (anytype pk, anytype pn)
{
    <do-Teil Zustand s3>          // durch Aktivitätendiagramm
                                // definiert und eingebettet
    if (<Bedingung 5>)           // internes bedingtes Ereignis
    {
        state = s2;
        <Aktion 5>
        Zustand_s2 ();
    }
};
```



Die Umsetzung in C++

Frage:

- Wann werden entry- und exit-Teile ausgeführt?
- Die Einbettung des entry-Teils kann durch Einfügen an den Anfang der Funktion `zustand_si()` erfolgen.
- Die Einbettung des exit-Teils kann durch Einfügen nach der case-Anweisung und einer evtl. abzurüfenden Bedingung stehen. Falls es sich um einen Zustandsübergang ohne Ereignis handelt, ist der exit-Teil nach der abzurüfenden Bedingung einzuschieben.



Die Umsetzung in C++

```
void Zustand_s3 (anytype pk, anytype pn)
{
    <entry-Teil>
    <do-Teil Zustand s3>      // durch Aktivitätendiagramm
                             // definiert und eingebettet
    if (<Bedingung 5>)       // internes bedingtes Ereignis
    {
        <exit-Teil des Zustandes s3>
        state = s2;
        <Aktion 5>
        Zustand_s2 ();
    }
};
```



Die Umsetzung in C++

```
virtual void Ereignis1 (anytype p1, anytype p2)
{
    switch (state)
    {
        case s1: if (<Bedingung 2>)
        {
            <exit-Teil des Zustandes s1>
            state = s3;
            <Aktion 2>
            Zustand_s3 (p1, p2);
        }
        break;
        case s3: if (<Bedingung 1>)
        {
            <exit-Teil des Zustandes s2>
            state = s1;
            <Aktion 1>
            Zustand_s1 (p1, p2);
        }
        break;
    } // switch
};
```



Codegenerierung Zustandsdiagramme

Andere sinnvolle Implementierungsmöglichkeiten:

- Design Pattern ´State´ [GHJV 96]
- Design Pattern ´State Table´ [Douglass 98]



Codegenerierung Zustandsdiagramme

Würden wir das oben beschriebene Vorgehen auch anwenden, wenn wir parallele Threads hätten?

- Das oben beschriebene Vorgehen sequenzialisiert den Ablauf und läuft in einem Thread ab.

Was passiert im oben beschriebenen Vorgehen, wenn mehrere asynchrone Nachrichten von einem Objekt ausgesandt werden?

- Eine Nachricht nach der anderen wird vollständig mit allen Unterverzweigungen abgearbeitet.
- Haben wir mehrere Threads zur Verfügung, könnten wir diese Threads parallel abarbeiten.
- Der Ablauf ist dann nicht mehr deterministisch.
- Die zuerst herein kommende Nachricht wird verarbeitet (oder Prioritätenvergabe u. ä.).

- **Parallelität**
- Bei mehreren parallel ablaufenden Threads darf zwischen Abfragen und Setzen des Attributes state kein anderer Thread das state-Attribut lesen und evtl. abhängig davon verändern.
- \Rightarrow erste Veränderung geht verloren!
- So eine Ausführung darf nur hintereinander geschehen:
- \Rightarrow Semaphore muss gesetzt werden.



Codegenerierung Zustandsdiagramme

Die Struktur einer Ereignismethode ändert sich wie folgt:

```
void ereignis (<parameterliste>
{
    derSemaphore.wait;                // Beginn des kritischen
    Abschnitts
    if (zustand_aktuell == state)
        if <Bedingung>
        {
            state = zustand_neu;
            <Aktion>
            derSemaphore.signal;        // Ende krit. Abschnitt
            <Zustandsmethode> (<parameterliste>);
        }
};
```

Das Kreieren des Semaphorobjekts erfolgt im Konstruktor der Klasse und wird mit dem Wert 1 initialisiert.