

Einführung in die Informatik II

Zustandsdiagramme

Prof. Bernd Brügge, Ph.D
Institut für Informatik
Technische Universität München

Sommersemester 2001

27. Juni - 2. Juli 2001

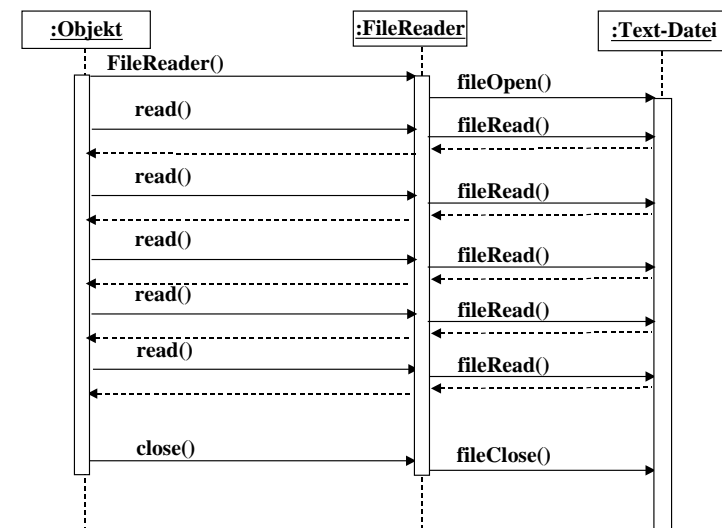
Ziele dieser Vorlesung

- ❖ Sie verstehen den Unterschied zwischen *Sequenzdiagrammen* und *Zustandsdiagrammen*
- ❖ Sie können für eine Klasse mit interessantem dynamischen Verhalten das zugehörige Zustandsdiagramm entwickeln.
- ❖ Sie verstehen die Beziehung zwischen einem Zustandsdiagramm und der assoziierten Klasse
- ❖ Sie kennen den Begriff der *Transition* und den Unterschied zwischen *Aktionen* und *Aktivitäten*
- ❖ Sie verstehen, dass verschiedene Benutzer oft verschiedene dynamische Modelle von Klassen haben
- ❖ Sie verstehen das Konzept *Modell-Sicht-Steuerung*

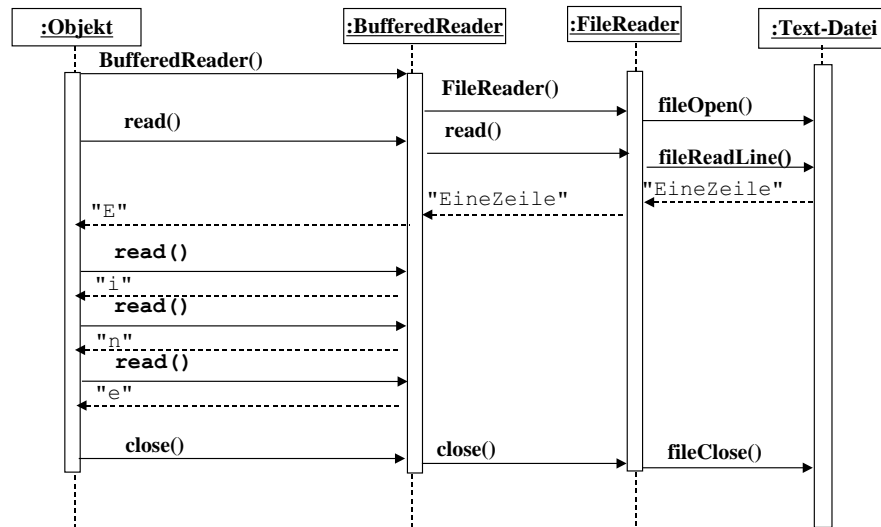
Modellierung eines Text-Editors

- ❖ Nehmen wir einmal den Anwendungsfall "*Schreiben einer Text-Datei*" aus der vorigen Vorlesung. Die Ereignisfolge besteht aus den folgenden Schritten:
 - ♦ Verbinde den Ausgabestrom mit einer Text-Datei.
 - ♦ Schreibe die einzelnen Daten auf den Strom (in einer Schleife).
 - ♦ Schließe den Strom.
- ❖ Wir können für diese Ereignisfolge ein Sequenzdiagramm erstellen, das die Interaktion mit dem Benutzer modelliert.
- ❖ Wir modellieren dabei zwei Fälle:
 - ♦ Ungepufferte Ausgabe
 - ♦ Gepufferte Ausgabe

Sequenzdiagramm für ungepufferte Text-Eingabe



Sequenzdiagramm für gepufferte Text-Eingabe



Copyright 2001 Bernd Brügge

Einführung in die Informatik II TUM Sommersemester 2001

13:52:29 6

Restriktionen beim dynamischen Verhalten von Objekten

- ❖ Ein Objekt vom **BufferedReader** oder **FileReader** stellt also Dienste zum Lesen von Daten bereit:
 - ◆ **Open ()**: Öffnen der Verbindung mit einer Datei
 - ◆ **Read ()**: Lesen eines Elementes.
 - ◆ **Close ()**: Schließen der Verbindung mit der Datei.
- ❖ Beim Gebrauch dieser Dienste gibt es allerdings einige Restriktionen:
 - ◆ **Read ()** darf nur aufgerufen werden, wenn vorher **Open()** aufgerufen wurde.
 - ◆ **Open ()** darf nicht zweimal hintereinander aufgerufen werden.
 - ◆ Wenn eine **Close ()**-Operation aufgerufen wurde, kann man **Read ()** danach nicht mehr aufrufen.
- ❖ Wie können wir diese Restriktionen modellieren?

Copyright 2001 Bernd Brügge

Einführung in die Informatik II TUM Sommersemester 2001

13:52:29 7

Modellierung des Verhaltens eines einzelnen Objektes

- ❖ *Aus der Sicht eines Stroms* ist es uninteressant, welche Objekte die Aufrufe zum Öffnen, Lesen, Schreiben und Schließen ausführen.
 - ◆ Die Aufrufe sind oft das Ergebnis der Abarbeitung unterschiedlicher Schritte in verschiedenen Anwendungsfällen.
- ❖ Der Strom muss nur sicherstellen, dass die Aufrufe in einer gewissen Reihenfolge stattfinden:
 - ◆ Der Strom muss erst geöffnet werden
 - ◆ Erst dann kann man schreiben oder lesen.
 - ◆ Am Ende muss der Strom geschlossen werden.
- ❖ Um diese Einschränkungen zu beschreiben, benutzen wir ein Zustandsdiagramm

Copyright 2001 Bernd Brügge

Einführung in die Informatik II TUM Sommersemester 2001

13:52:29 8

UML-Zustandsdiagramm

- ❖ **UML-Zustandsdiagramm**: Eine Notation zur Modellierung des dynamischen Verhaltens eines einzelnen Objektes.
- ❖ Ein Zustandsdiagramm besteht aus einer Anzahl von
 - ◆ **Zuständen**, die das Objekt annehmen kann.
 - ◆ **Transitionen**, die den Übergang zwischen den Zuständen regulieren.
 - ◆ **Operationen**, die in einem Zustand oder während einer Transition ausgeführt werden.

Copyright 2001 Bernd Brügge

Einführung in die Informatik II TUM Sommersemester 2001

13:52:29 9

Zustände

- ❖ Der **Zustand eines Objektes** ist eine Funktion $Z: A_1 \times A_2 \times \dots \times A_n \rightarrow \mathbb{N}$, wobei A_1, A_2, \dots, A_n die Trägermengen einiger oder aller Attribute der zugehörigen Klasse sind.
- ❖ **Beispiel:** Ich möchte ein Bank-Konto eröffnen. Ich will wissen, ob ich der Bank mein Geld anvertrauen kann. Die Bank kann in einem von zwei Zuständen sein:

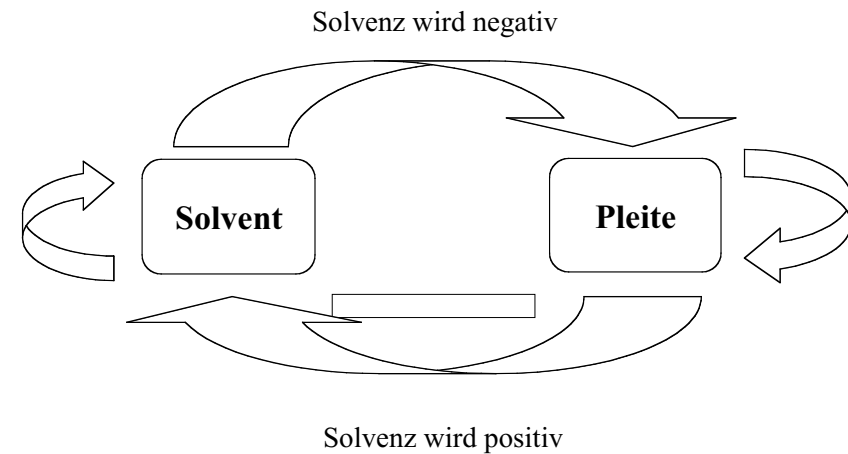
- ♦ **Pleite:** Die Bank ist bankrott
- ♦ **Solvent:** Die Bank hat Geld

Solvenz: $\text{Verfügbares Geld} \times \text{Grundstücke} \times \text{Hypotheken} \rightarrow \{\text{Pleite}, \text{Solvent}\}$

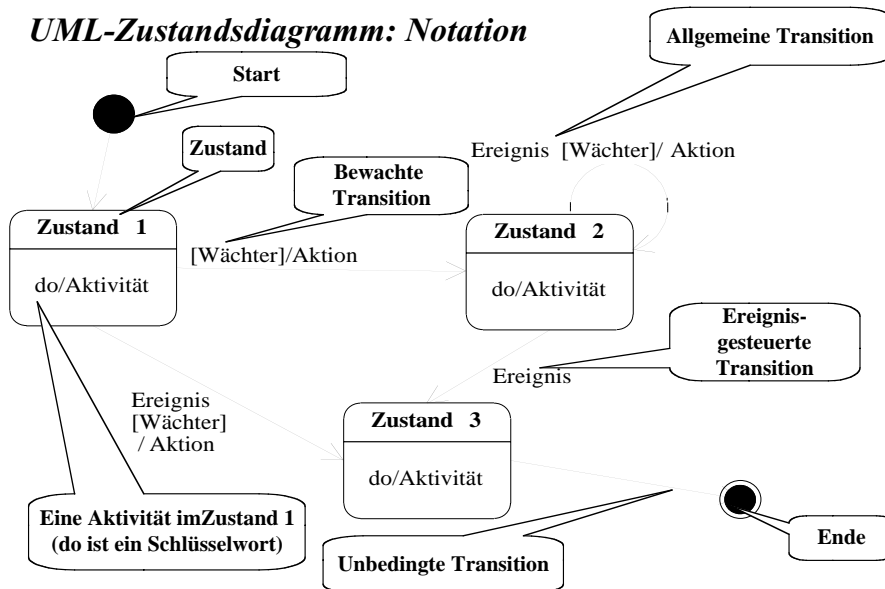
Bank
Konten
Verfügbares Geld
Grundstücke
Hypotheken
Filialen

- ❖ Den Wert einer Zustandsfunktion bei gegebenen Attribut-Werten bezeichnen wir als den **aktuellen Zustand** des Objektes.

Zustandsdiagramm für die Bank



UML-Zustandsdiagramm: Notation



Transitionen

- ❖ **Definition Transition:** Eine Relation zwischen zwei Zuständen Z_1 und Z_2 eines Objekts in einem Zustandsdiagramm (Spezialfall: $Z_1 = Z_2$). Die Transition (Z_1, Z_2) beschreibt den *Übergang* des Objekts vom Zustand Z_1 in den Zustand Z_2 .
- ❖ Eine Transition kann eine **Transitionsbeschriftung** haben, die aus bis zu drei optionalen Teilen besteht:
 - ♦ **Ereignis:** Ein Ereignis aus dem Ereignismodell.
 - ♦ **[Wächter]:** Ein prädikatenlogischer Ausdruck.
 - ♦ **/Aktion:** Eine kurzzeitige, nicht unterbrechbare Operation, die bei der Ausführung der Transition ausgeführt wird.
- ❖ Allgemeine Form der Beschriftung in UML-Zustandsdiagrammen:

***Ereignis*[Wächter]/Aktion**

Arten von Transitionen

❖ Je nachdem, welche optionale Teile nicht erwähnt sind, unterscheiden wir folgende Transitionen:

♦ **Ereignis-gesteuerte Transition:**

Die Transition wird durch ein Ereignis ausgelöst.

Beschriftung: **Ereignis/Aktion** oder **Ereignis**

♦ **Bewachte Transition:**

Die Transition wird durch Wahrwerden des Wächter-Prädikates ausgelöst.

Beschriftung: **[Wächter]/Aktion** oder **[Wächter]**

♦ **Unbedingte Transition:**

Die Transition wird durch die Beendigung der Aktivität des aktuellen Zustands ausgelöst.

Beschriftung: **/Aktion** oder keine Beschriftung

Operationen: Aktivitäten und Aktionen

❖ In Zuständen und bei den Übergängen zwischen Zuständen können **Operationen** ausgeführt werden.

♦ **Aktivität:** Eine Operation, die ausgeführt wird, wenn sich das Objekt in einem bestimmten Zustand befindet.

♦ **Aktion:** Eine Operation, die während der Transition von einem Zustand zum anderen ausgeführt wird.

❖ Gemeinsam ist beiden Arten von Operationen, dass sie bei der Programmierung als Methoden implementiert werden. Es gibt aber einen Unterschied:

♦ Aktionen sind im allgemeinen kurz und nicht durch Ereignisse unterbrechbar.

♦ Aktivitäten können länger dauern und sind durch Ereignisse unterbrechbar.

Modellierung mit Zustandsdiagrammen: Beispiele

❖ Wir modellieren jetzt die Zustände von zwei Beispiel-Objekten, um ein bisschen Übung mit Zustandsdiagrammen zu bekommen:

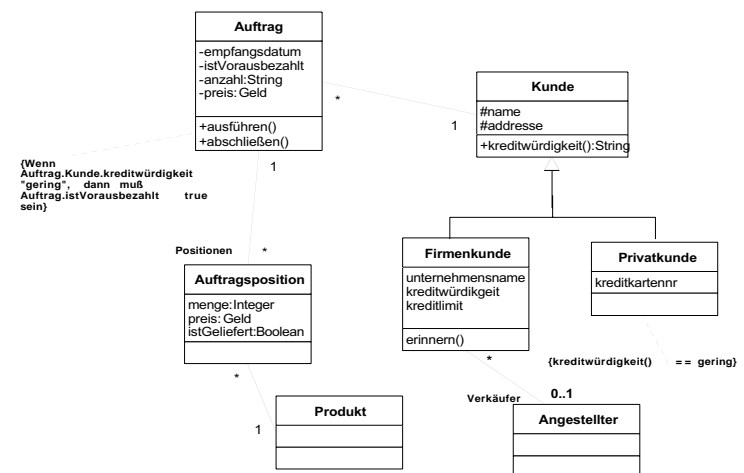
1. Modellierung eines Auftrages

♦ Ausgangspunkt: Klassendiagramm für einen Auftrag (siehe Info II - Vorlesung 1 (UML))

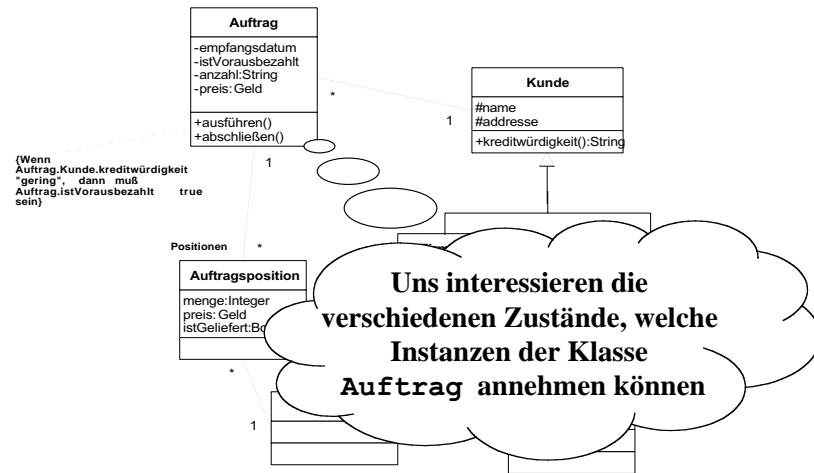
2. Modellierung eines Text-Editors

♦ Ausgangspunkt: Text-Editor (siehe Info II - Vorlesung 7 (Ströme und Dateien))

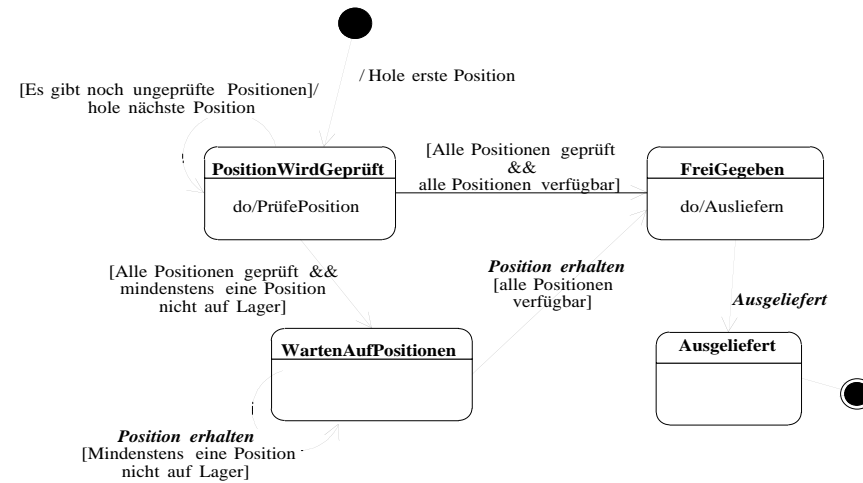
1. Beispiel: Zustandsmodellierung eines Auftrages



1. Beispiel: Zustandsmodellierung eines Auftrages



Zustandsdiagramm für Auftrag



Transitionen aus dem Zustand *PositionWirdGeprüft*

- ❖ Wenn wir in Zustand **PositionWirdGeprüft** sind, gibt es 3 Situationen:
 - ◆ Wenn noch nicht alle Auftragspositionen geprüft worden sind, holen wir die nächste Position, bleiben aber im Zustand **PositionWirdGeprüft**.
 - ◆ Wenn alle Auftragspositionen geprüft sind, und alle auf Lager sind, dann gehen wir in den Zustand **Freigegeben**.
 - ◆ Wenn alle Auftragspositionen geprüft, aber nicht alle vorrätig sind, gehen wir in den Zustand **WartenAufPositionen** über.

Der Zustand *WartenAufPositionen*

- ❖ In diesem Zustand gibt es keine Aktivitäten. Es ist ein sogenannter **Wartezustand**:
Wir warten auf ein Ereignis **Position Erhalten**.
 - ◆ Wenn das Ereignis kommt, überprüfen wir die Wächterbedingung der beiden Transitionen, die von dem Zustand wegführen.
 - ◆ Wenn das Prädikat [**Alle Positionen verfügbar**] wahr ist, gehen wir in den Zustand **Freigegeben**.
 - ◆ Wenn das Prädikat [**Minderstens eine Position nicht auf Lager**] wahr ist, bleiben wir im Zustand **WartenAufPositionen**
- ❖ Was ist, wenn das Ereignis **Position Erhalten** nie kommt?

Zustandsdiagramme führen zu revidierten Klassendiagrammen

- Die genaue Modellierung der Zustände führt oft zu Modifikationen oder Korrekturen der assoziierten Klasse

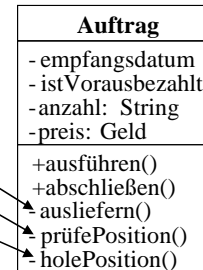
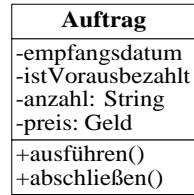
Die Aktivitäten im Zustandsdiagramm

do/Ausliefern
do/PrüfePosition

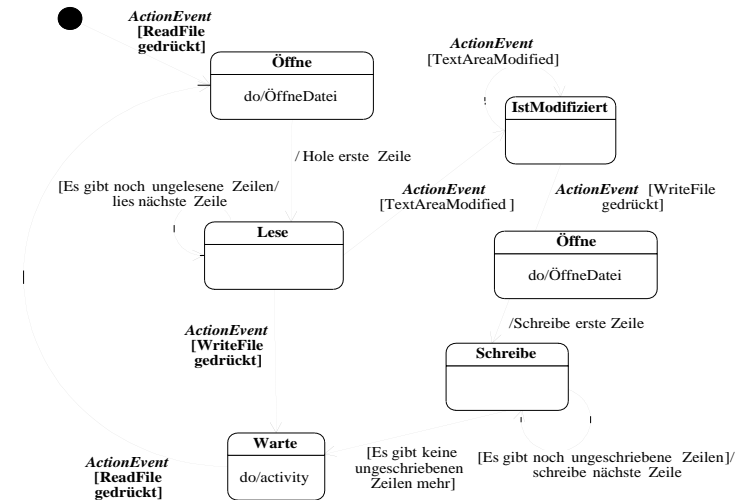
und die Aktionen im Zustandsdiagramm

/hole erste Position
/hole nächste Position

ergeben neue Operationen in der Klasse

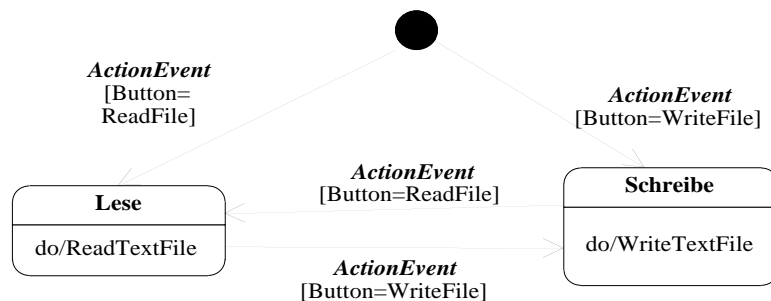


2. Beispiel: Zustandsmodellierung des Text-Editors

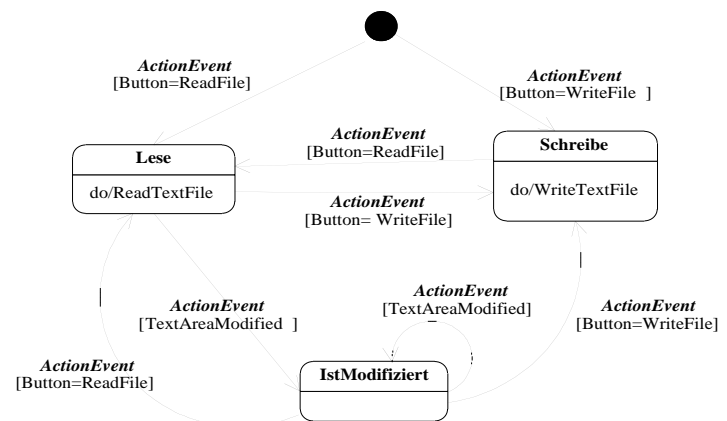


Ist dies ein gutes Modell?

Ein besseres Modell für den Text-Editor



Ein revidiertes Modell des Text-Editors



Wahl von Wächterbedingungen

- ❖ Aus einem Zustand kann immer nur genau **eine** Transition gewählt werden, auch wenn der Zustand mehrere Transitionen zu anderen Zuständen zulässt.
 - ◆ Der Zustand **IstModifiziert** erlaubt 3 Transitionen:
 - ◆ zum Zustand **Lese**, wenn ein **ActionEvent** eintritt, weil der **Knopf ReadFile gedrückt** wurde.
 - ◆ zum Zustand **Schreibe**, wenn ein **ActionEvent** eintritt, weil der **Knopf WriteFile gedrückt** wurde.
 - ◆ zu sich selbst, wenn ein **ActionEvent** eintritt, weil **Text im Textbereich verändert** wurde.
- ❖ **Heuristik:** Man sollte Wächterbedingungen so wählen, dass sie sich bei einem gegebenen Ereignis in einem bestimmten Zustand gegenseitig ausschließen.

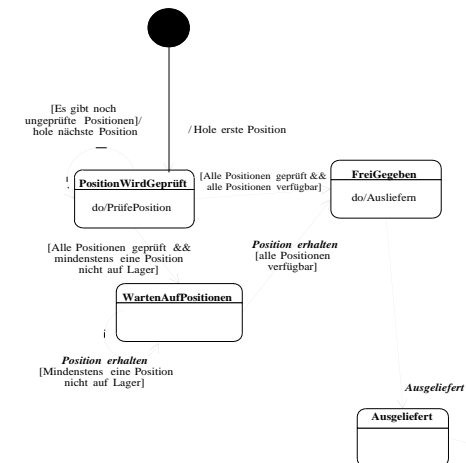
Zustände und Ereignis-basierte Programmierung

- ❖ In der Ereignis-basierten Programmierung ist es ein häufiger Modellierungsfehler, für bestimmte Zustände einige Ereignisse aus dem Ereignismodell unberücksichtigt zu lassen.
 - ◆ Beispiel: Wir wollen in jedem Zustand das Programm abbrechen können.
- ❖ Während der Implementierung ist es sehr schwierig, einen derartigen Unterlassungsfehler zu finden.
 - ◆ Das Informatik-System reagiert nicht auf das Abbruchkommando, und Sie wissen nicht, warum.
- ❖ Überprüfen Sie für alle Zustände Z_i und alle Ereignisse E_j aus dem Ereignismodell, ob E_j im Zustand Z_i vorkommen kann. Wenn ja, tragen Sie eine mit E_j beschriftete Transition für Z_i ein.
- ❖ Allerdings wird durch zu viele Transitionen ein Zustandsdiagramm schnell unleserlich.

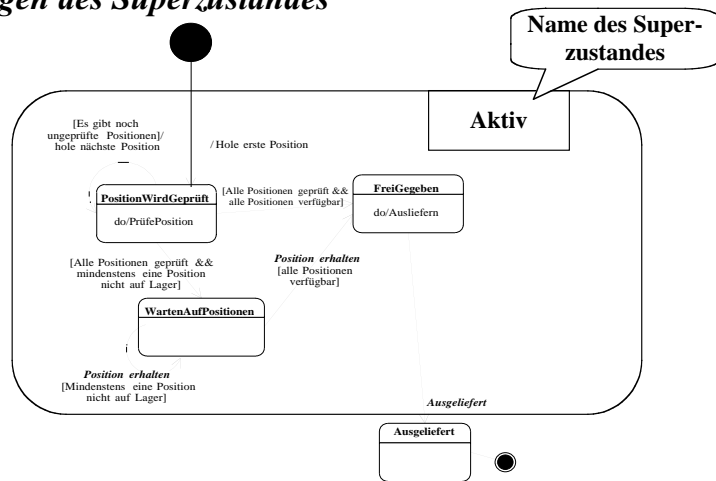
Hierarchische Zustände

- ❖ Wir wollen einen Auftrag zu jedem Zeitpunkt vor seiner Auslieferung abrechnen können.
- ❖ Dies können wir modellieren, in dem wir für das Ereignis "**Abbruch**" separate Transitionen von jedem der Zustände "**PositionWirdGeprüft**", "**Freigegeben**" und "**WartenAufPositionen**" zum Anfangszustand einführen.
- ❖ Eine bessere Alternative ist es, für die drei Zustände einen sog. **Superzustand** (engl. super state) zu erzeugen, und von diesem eine einzelne Transition zum Anfangszustand eintragen.
- ❖ Die verschachtelten Unterzustände *erben* einfach alle Transitionen des Superzustandes.

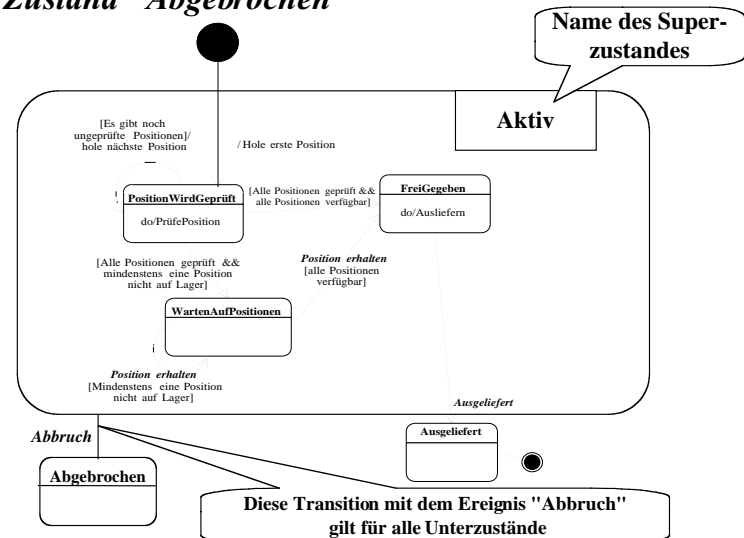
Modellierung mit Superzuständen (1): Neuzeichnen der alten Zustände



Modellierung mit Superzuständen (2): Hinzufügen des Superzustandes

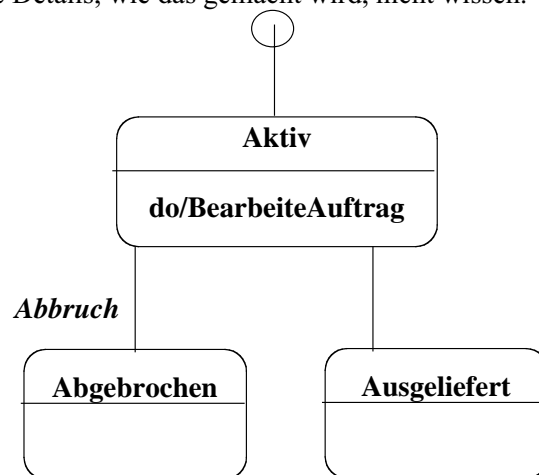


Modellierung mit Superzuständen (3): Neuer Zustand "Abgebrochen"



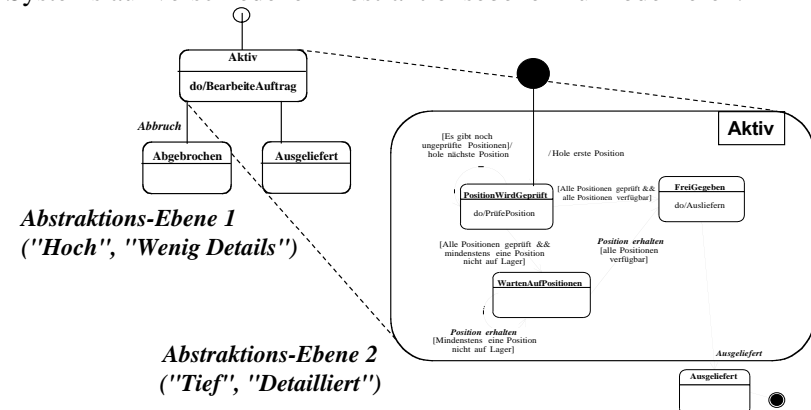
Der Superzustand als "schwarzer Kasten" (black box)

- ❖ Im Zustand **Aktiv** wird der Auftrag bearbeitet.
Wir wollen die Details, wie das gemacht wird, nicht wissen.



Abstraktionsebenen in Zustandsdiagrammen

Superzustände erlauben es uns, die dynamischen Eigenschaften eines Systems auf verschiedenen Abstraktionsebenen zu modellieren.



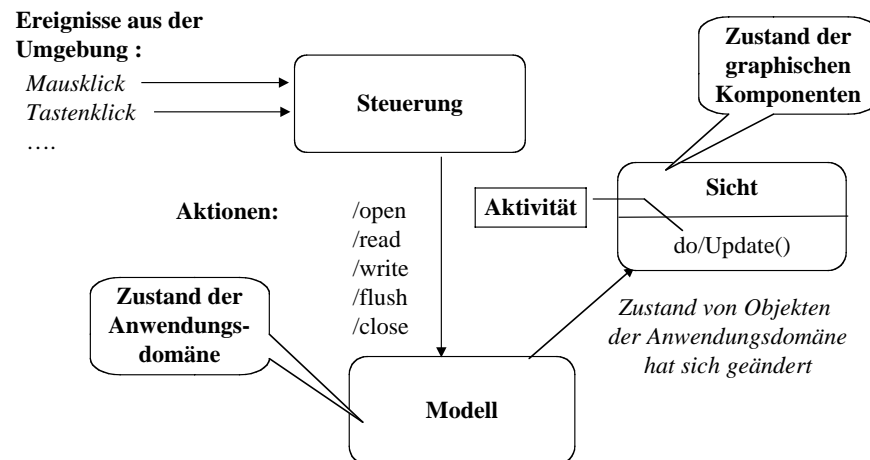
Modell-Sicht-Steuerung: Ein Konzept für die Strukturierung offener Systeme

- ❖ Wir führen jetzt mit **Modell-Sicht-Steuerung** ein Entwurfs-Konzept ein, das bei der Entwicklung offener Systeme (vor allem interaktive Systeme mit graphischen Benutzeroberflächen) sehr nützlich ist.
- ❖ **Modell-Sicht-Steuerung** (MSS, engl. Model-View-Controller (MVC)) beschreibt eine System-Struktur mit drei Subsystemen:
 - ♦ **Modell (model)**: Subsystem zur *Modellierung der Anwendungsdomäne*
 - ♦ **Sicht (view)**: Subsystem zur *Realisierung einer Bedienoberfläche*
 - ❏ **Steuerung (controller)**: Subsystem zur *Reaktion auf Ereignisse* aus der Umgebung.
- ❖ Dieses Konzept ist komplizierter als die bisher eingeführten Entwurfsmuster.
 - ♦ Schach-Analogie: Die bisherigen Muster beschrieben "*Endspiele*" mit wenigen Komponenten. Modell-Sicht-Steuerung beschreibt eine "*Mittelspiel*"-Situation mit vielen Komponenten.

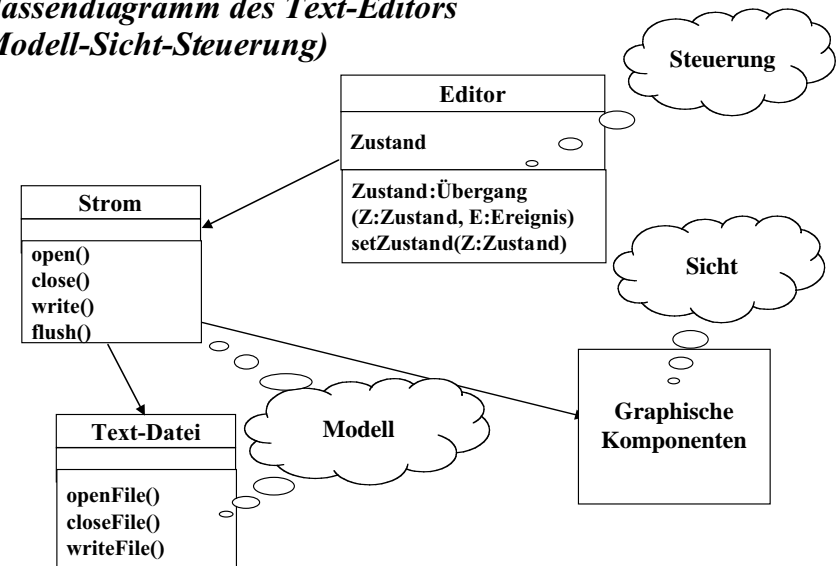
Modell-Sicht-Steuerung: Subsystem-Kooperation

- ❖ Die **Modell-** und **Sicht-**Subsysteme kooperieren nach dem bekannten Beobachtermuster:
 - ♦ Wenn sich der Zustand von Objekten der Anwendungsdomäne ändert, wird die **Update ()**-Methode auf den angeschlossenen Sichten aufgerufen, damit alle Sichten den neuen Zustand reflektieren.
- ❖ Das **Steuerungs**-Subsystem ist neu:
 - ♦ Es entkoppelt das Ereignis-Modell vollständig von dem Rest des Systems.
 - ♦ Es reagiert auf Ereignisse *aus der Umgebung* (z.B. vom Benutzer) und sendet Ereignisse zum Anwendungs-Modell oder ruft öffentliche Operationen von Komponenten des Anwendungs-Modells auf.
- ❖ Ein Text-Editor mit graphischer Bedienoberfläche lässt sich gut mit Modell-Sicht-Steuerung modellieren.

Entwurf eines Text-Editors mit Modell-Sicht-Steuerung

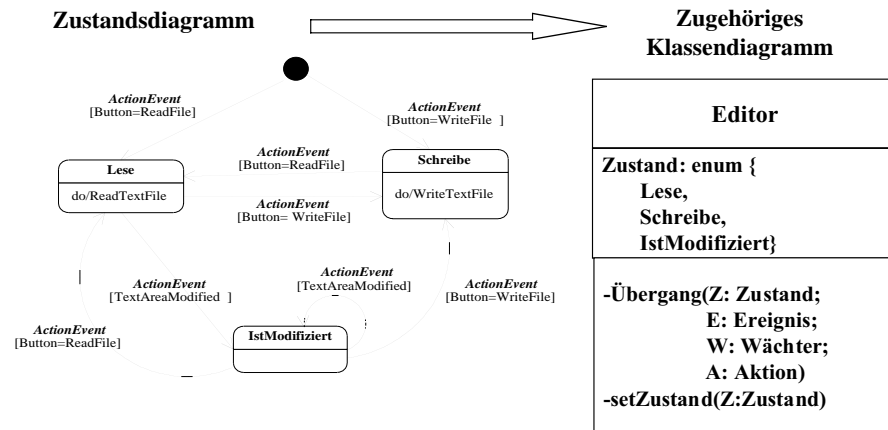


Klassendiagramm des Text-Editors (Modell-Sicht-Steuerung)



Detaillierter Entwurf

- ❖ Wir betrachten nur das Modell für die Steuerung:



Modellierung von Informatik-Systemen: Übersicht

- ❖ Mit der Einführung von Zustandsdiagrammen haben Sie nun ein ansehnliches Arsenal von Notationen, um die wesentlichen Eigenschaften von Informatik-Systemen zu modellieren:

- ♦ **Funktionale Eigenschaften:**

- ♦ Szenarios und Anwendungsfalldiagramme

- ♦ **Dynamische Eigenschaften:**

- ♦ Sequenzdiagramme und Zustandsdiagramme

- ♦ **Statische Eigenschaften:**

- ♦ Klassendiagramme und Instanzdiagramme

- ❖ Die Modelle "bereichern" sich gegenseitig:

- ♦ Aktionen und Aktivitäten in Zustandsdiagrammen sind Kandidaten für Anwendungsfälle oder Operationen in Klassendiagrammen
 - ♦ Attribute in Klassen sind Kandidaten für Zustände

Zusammenfassung

- ❖ Ein **UML-Zustandsdiagramm** modelliert das dynamische Verhalten eines **einzelnen** Objekts in **verschiedenen** Anwendungsfällen. Es ist mit **einer** Klasse assoziiert.
- ❖ Ein **Zustand** ist die Abbildung von **Objektattributen** auf einen Wert. Innerhalb eines Zustands können mehrere **Aktivitäten** stattfinden.
- ❖ **Transitionen** sind Übergänge zwischen Zuständen und haben bis zu drei Komponenten: **Ereignis, Wächter, Aktion**
- ❖ Aktionen sind kurzzeitige Operationen bei Transitionen, Aktivitäten sind längerdauernde Operationen in Zuständen.
 - ♦ **Aktionen und Aktivitäten** sind deshalb gute Kandidaten für Operationen in der assoziierten Klasse.
- ❖ Vermeiden Sie "Spaghetti"-Modelle! Verwenden Sie **Superzustände**, um die Lesbarkeit von Zustandsdiagrammen zu erhöhen.
- ❖ Das Konzept **Modell-Sicht-Steuerung** führt zu einfacher erweiterbaren und wiederverwendbaren Strukturen bei offenen Informatik-Systemen.
 - ♦ Die **Steuerung** reagiert auf Ereignisse aus der Umgebung.