4.
Softwarearchitektur und Softwareentwurf

4.3
Architektur von eingebetteten und Echtzeitsystemen

Literatur: Sommerville 13
Balzert LE 31
Jalote 6

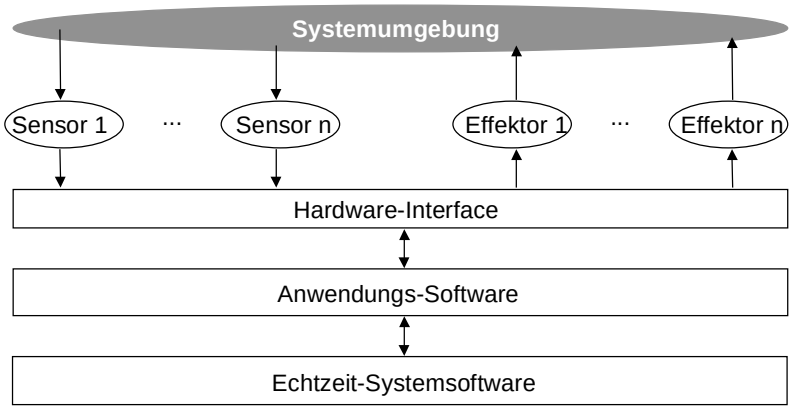
Echtzeit-Systeme

- *Echtzeit-System bzw. Realzeit-System (real time system):*
  - Korrektes Funktionieren für eine gegebene Eingabe definiert als:
    - » richtige Ausgabe **und rechtzeitige** Ausgabe
  - Zeitbegrenzung in "echter" Zeit, d.h. nicht relativ zu systeminternen Maßgrößen
- *"harte" Echtzeitanforderungen:*
  - Funktion verfehlt bei Nichteinhalten der Zeitbedingung (z.B. Flugzeugsteuerung, Ampelsteuerung)
- *"weiche" Echtzeitanforderungen:*
  - Funktion verschlechtert bei Nichteinhalten der Zeitbedingung (z.B. Telefonvermittlung, Videopumpe)
- *"feste" Echtzeitanforderungen:*
  - Kombination aus weicher Echtzeitanforderung und harter Schranke (z.B. Beatmungsmaschine)

Besonderheiten von Echtzeitsystemen

- Oft: Hardware- und Software-System oder *embedded system*
  - Spezielle Peripheriegeräte, spezielle Hardwarearchitektur
- Reaktivität
  - Bereitschaft zur Reaktion auf Ereignisse, die in nicht vorhersehbarer Reihenfolge und zu nicht vorhersehbarer Zeit erfolgen
- Nebenläufigkeit *(concurrency)*
  - Gleichzeitigkeit von Ereignissen der Umgebung
  - Interne Nebenläufigkeit des Systems
- Verteilung *(distribution)*
  - Manchmal durch die Anwendung unvermeidbar (z.B. Telekommunikation)
  - Hilfsmittel zur Erreichung von schneller Reaktion und hoher Zuverlässigkeit (Redundanz)
- Dynamische Anpassung an die Umgebung
  - z.B. Umbau einer rechnergesteuerten Produktionsanlage

Realzeitsystem: Grundsätzliche Architektur



- Oft sehr kleine ergänzbare "microkernel" (z.B. 8 kByte)
  Beispiele: OS-9, Chorus, QNX, LynxOS, VxWorks

Softwareentwurf für Echtzeitsysteme und eingebettete Systeme

- Weitgehend vorgegebene physikalische Sicht der Architektur
  - Architekturdiskussion in der Analyse
- Berücksichtigung gegebener Architektur im Entwurf
  - Hardwareschnittstellen
  - Betriebssystemschnittstellen
  - Kommunikation und Verteilung
- Modellierung verwendbar für praktische Realisierung
  - Relativ kleine Differenz Analysemodell-Entwurfsmodell-Implementierung
  - Gute Werkzeugunterstützung
  - Effektive Codegenerierung aus Modellen möglich
- Deshalb: Diskussion von weiteren (implementierungsnahen) Modellierungssprachen

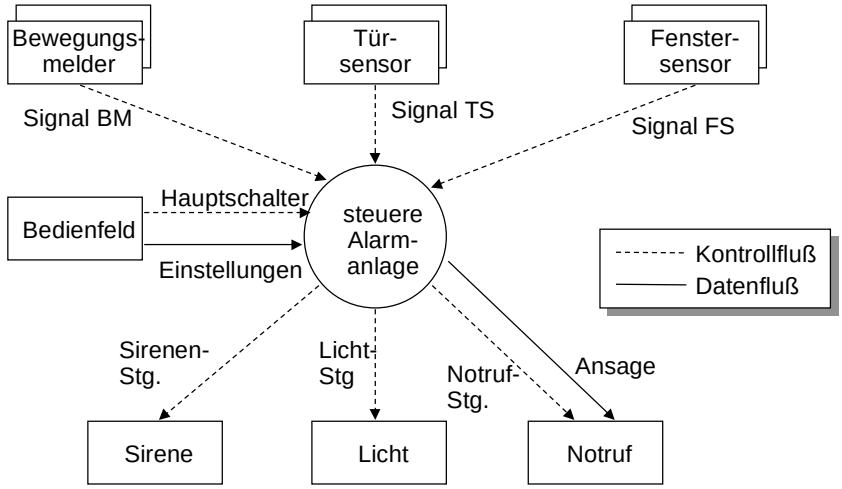
Entwicklungsmethoden für Echtzeitsysteme

- Aufbauend auf Strukturierter Analyse:
  - Structured Analysis/Real Time (SA/RT) – "Industriestandard"
    - » Ward/Mellor 1985, Hatley/Pirbhai 1987
- Aufbauend auf Objektorientierter Analyse:
  - Real-Time Object-Oriented Modeling (ROOM)
    - » Selic/Gullekson/Ward 1994
  - UML for Real-Time Systems
    - » Selic/Rumbaugh 1998
- Aufbauend auf Programmablaufplänen (Kontrollflußdiagrammen):
  - System Design Language (SDL)
    - » Standard in der Telekommunikation
    - » Derzeit in Integration mit objektorientierten Ansätzen (SDL-2000)

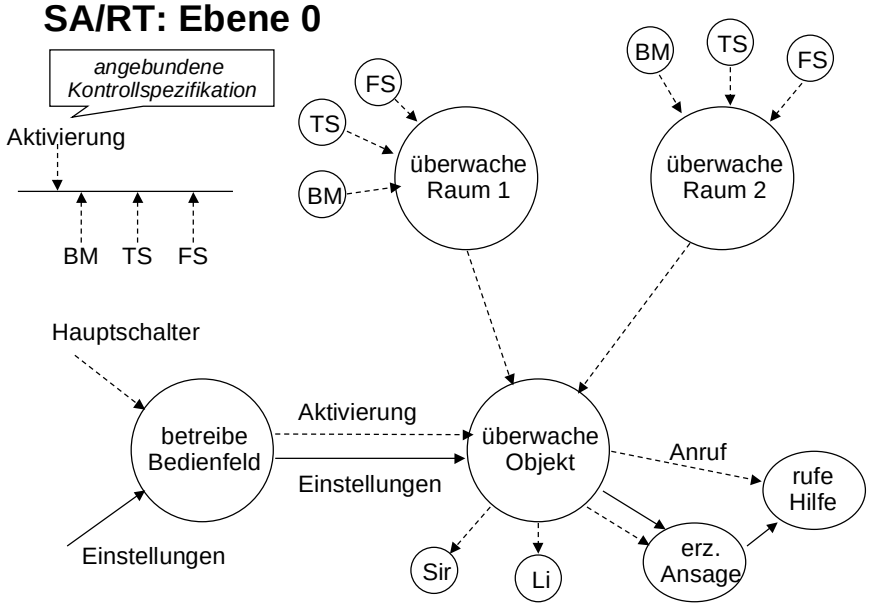
Beispiel: Alarmanlage

- Zu überwachen sind mehrere (im Beispiel 2) Räume
- Jeder Raum hat (im Beispiel):
  1 Bewegungsmelder, 1 Türsensor, 1 Fenstersensor
- Die Anlage wird über eine Schalteinrichtung aktiviert und deaktiviert.
- Nach Aktivierung gelten drei Minuten Vorlauf, bevor die Anlage "scharf" geschaltet wird.
- Reaktionszeit für Alarm: 1/2 Minute
- Bei Alarm ist folgendes zu tun:
  - Notruf absetzen (Sprachsynthese)
  - Licht einschalten
  - Sirene auslösen

SA/RT: Kontextdiagramm

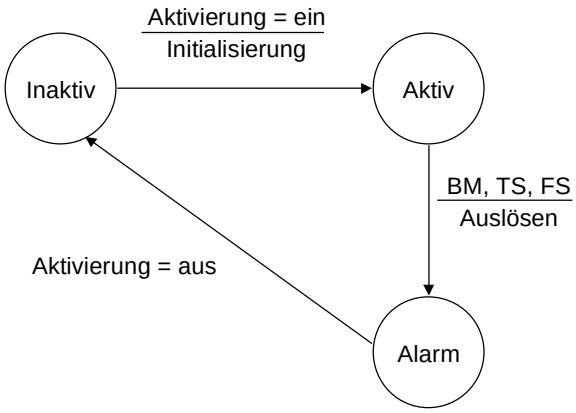


SA/RT: Ebene 0



SA/RT: Kontrollspezifikations - Automat

- Kontrollspezifikation:
  - besteht aus Zustandsautomat und/oder Entscheidungstabelle
  - wird an das Flußdiagramm durch Balkennotation angebunden



SA/RT: Kontrollspezifikations - Tabelle

Steuer-aktion aus Automat \ Prozeß	überw. Raum 1	überw. Raum 2	Sir.	Licht	erz. Ansage	rufe Hilfe
Initialisierung	1	2	0	0	0	0
Auslösen	0	0	4	3	1	2

- Im allgemeinen: Entscheidungstabelle
- Bedeutung: Prozeßaktivierungstabelle
- Einträge: Reihenfolge der Aktivierung

SA/RT: Zeitspezifikationen

Eingabesignal	Ereignis	Ausgabesignal	Ereignis	Zeitbedingung
Hauptschalter	ein	Aktivierung	ein	180 sec
BM, TS, FS	ein	Anruf	wählen	< 30 sec

- Weitere Zeitbedingungen (z.B. Frequenz einer Sensor-Abfrage) im Data Dictionary (Requirements Dictionary)

SA/RT: Zusammenfassung

- Ausgereifte, aber komplexe Darstellungsformen
  - Zusammenspiel verschiedener Diagramme
  - Nicht einfach zu verstehen
- Methodische Unterstützung vorhanden
- Relativ geringe Unterschiede zwischen Analyse, Entwurf und Implementierung
  - Spezifikation = abstrakte Sicht auf Implementierung
  - Simulation von Spezifikationen
- Konsistenzbedingungen sehr komplex
- Keine Daten- oder Verhaltenskapselung
- Schlechte Unterstützung für mehrfache Instanzen desselben Verhaltens

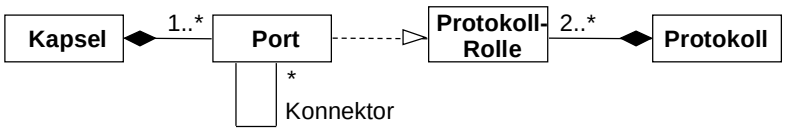
Objektorientierte Realzeit-Modellierung

- Verfeinerte Zustandsmodellierung:
  - Shlaer/Mellor 1992 ("Modelling the World in States")
- Einführung zusätzlicher Strukturelemente ("Aktoren")
  - Selic/Gullekson/Ward 1994
    (ROOM - Real-Time Object-Oriented Modelling)
- Realzeit-Modellierung mit "purem" UML
  - Douglass (1998, 1999)
    (Angelehnt am Werkzeug "Rhapsody" der Firma i-Logix)
- Von ROOM zu UML/ROOM:
  - ROOM-Werkzeug: ObjecTime Ltd. (Kanada), B. Selic et al.
  - 1998: "White Paper" von B. Selic (ObjecTime)/J. Rumbaugh (Rational)
  - 1999: Fusion von ObjecTime und Rational
    - » ObjecTime-Werkzeug nun spezielle Version von Rose (Rose-RT)
- Entscheidung über "offizielle" Realzeit-Version von UML noch offen.

Sprachelemente von ROOM/UML

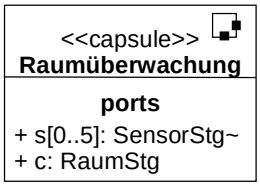
- ROOM/UML gibt zusätzlich zu den UML-Sprachelementen bewährte Architekturmuster für Echtzeitsysteme mit einer einfachen Kurznotation vor.
- Kapsel *(capsule)*: (in ROOM: Aktor *(actor)*)
  - Komplexes, möglicherweise verteiltes physikalisches Objekt, das mit seiner Umgebung über Signale kommuniziert.
- Port:
  - Physikalischer Teil der Implementierung einer Kapsel, über den die Kapsel mit der Außenwelt durch Austausch von Signalen kommuniziert.
  - Assoziiert mit dem Port ist ein *Protokoll*, das die über den Port ausgetauschten Informationen beschreibt.
- Konnektor *(connector)*: (in ROOM: Bindung *(binding)*)
  - Abstrakte Sicht auf einen Kommunikationskanal, der zwei oder mehr Ports verbindet.
  - Partner-Ports eines Konnektors müssen aufeinander abgestimmte Protokolle benutzen (*komplementäre Protokoll-Rollen*)

Metamodell für Kapseln

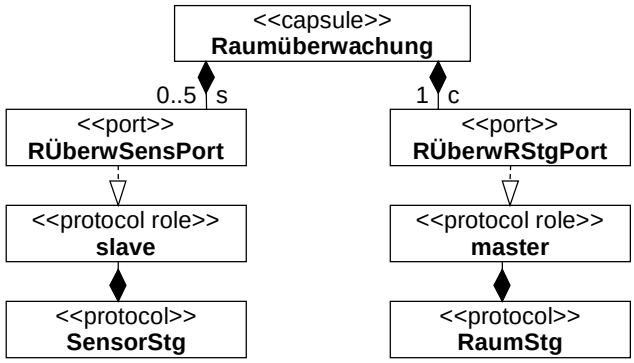


- Alle aufgeführten Klassen sind Spezialfälle des UML-Konstrukts "Klasse".
  - Stereotypen <<capsule>>, <<port>>, <<protocol>>, <<protocolRole>>
- Für Port-, Protokoll- und Protokoll-Rolle-Klassen werden i.a. keine individuellen Attribute spezifiziert
- Kapsel-Klassen enthalten private Attribute und Operationen.
- Protokolle definieren Signale (Operationen der Protokoll-Rollen) zur Kommunikation zwischen Kapseln.
- Die meisten Protokolle kennen zwei Standard-Rollen (z.B. *master* und *slave* genannt)

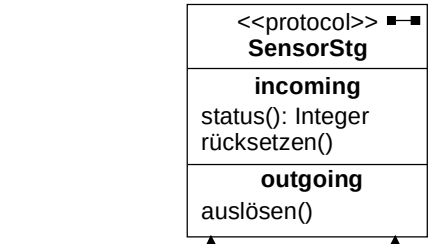
ROOM/UML-Notation für Kapsel-Klassen

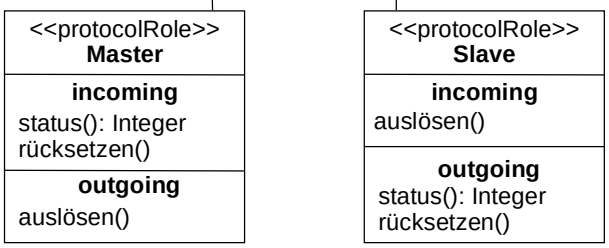


- Spezialisierung der UML-Klassennotation:
  - zusätzliches "compartment" (ports)
- Abkürzung für unten dargestelltes Klassendiagramm



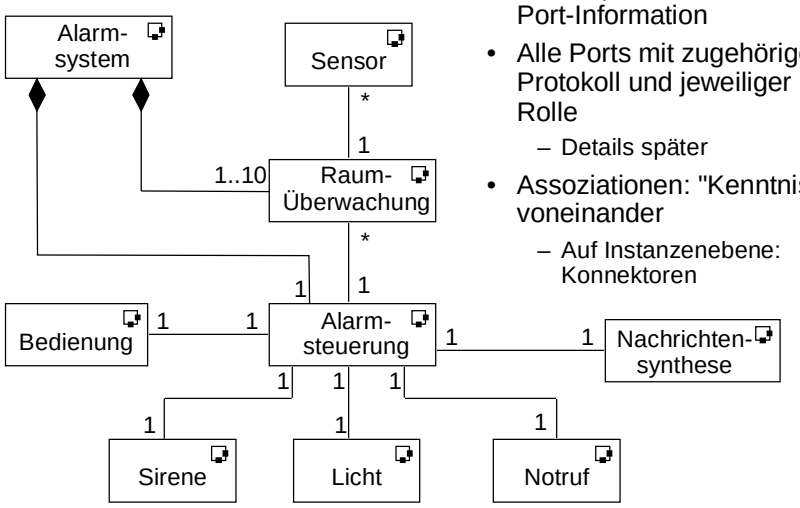
Protokoll-Spezifikation in ROOM/UML





- Zwei-Partner-Protokolle:
  - Master- und Slave-Rolle "konjugiert", d.h. ableitbar aus demGegenstück
  - Kurznotation:
    Protokollname (=Master)
    Protokollname~ (=Slave)
- Mehr-Partner-Protokolle:
  - selten

Beispiel: Klassendiagramm in ROOM/UML

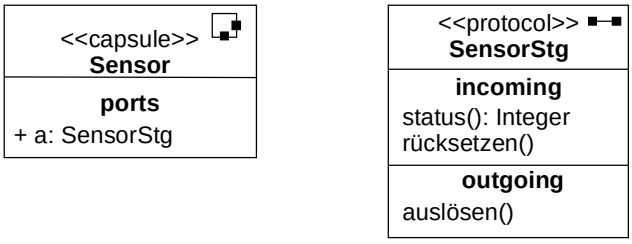


- Alle Kapsel-Klassen mit Port-Information
- Alle Ports mit zugehörigem Protokoll und jeweiliger Rolle
  - Details später
- Assoziationen: "Kenntnis" voneinander
  - Auf Instanzenebene: Konnektoren

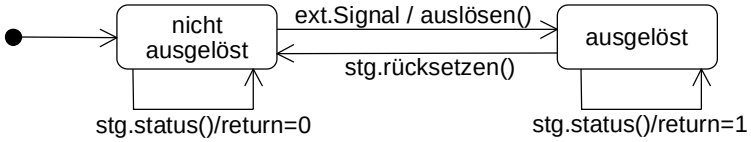
Spezifikation des Verhaltens einer Kapsel

- Zwei Möglichkeiten (je Port wählbar):
- Spezifikation durch Zustandsdiagramm
  - *end port*
  - Zur Kapsel assoziierte Zustandsmaschine behandelt alle Signale des dem Port zugeordneten Protokolls
- Verfeinerung durch enthaltene Kapseln
  - *relay port*
  - Port dient nur zum Weiterreichen von Signalen an Unterkapseln
  - Das Gesamtsystem kann als eine Kapsel verstanden werden
    - » Hierarchische Zerlegung des Gesamtsystems in Kapseln
  - In ROOM/UML beschrieben durch *Kollaboration* von *Instanzen* innerer Kapseln
    - » Spezialfall des UML-*Kollaborationsdiagramms*

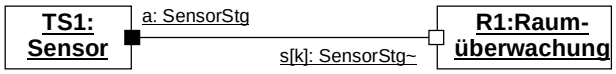
Einfaches Zustandsdiagramm für Kapsel



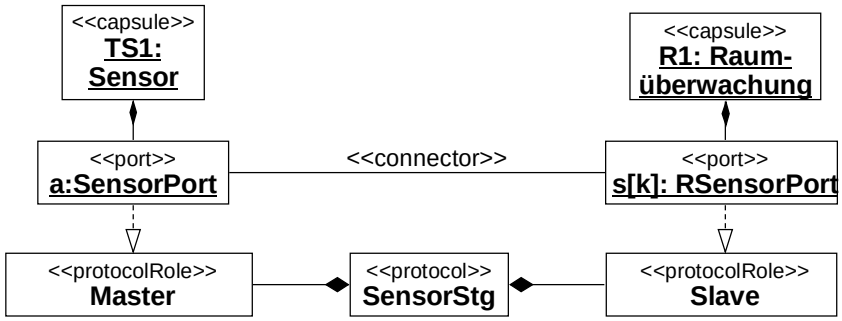
- Zugeordnetes Zustandsdiagramm für Kapsel Sensor:



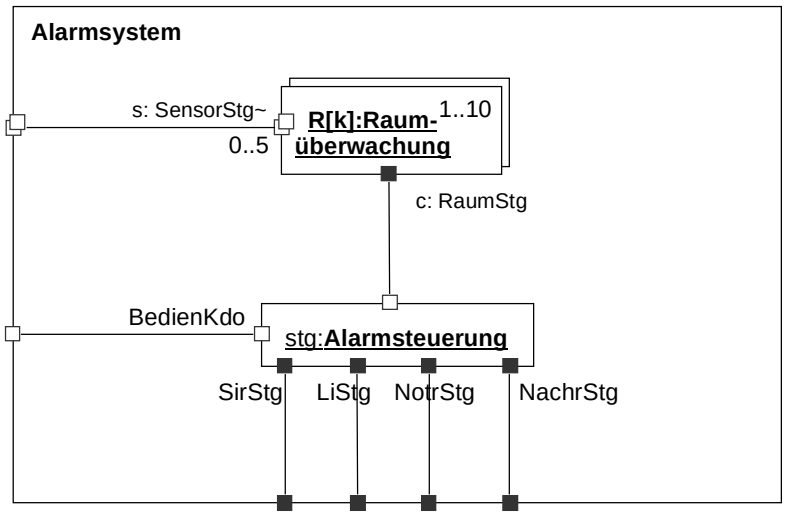
ROOM/UML Instanzendiagramm (Strukturdiagramm)



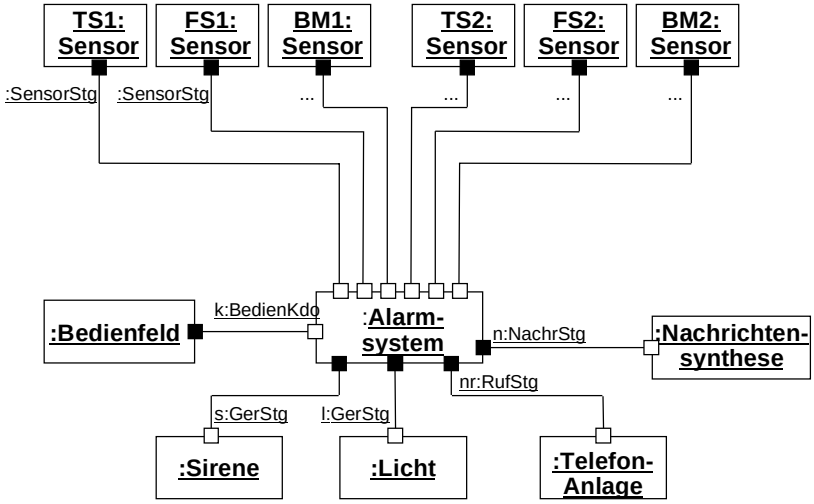
ist Kurzform für:



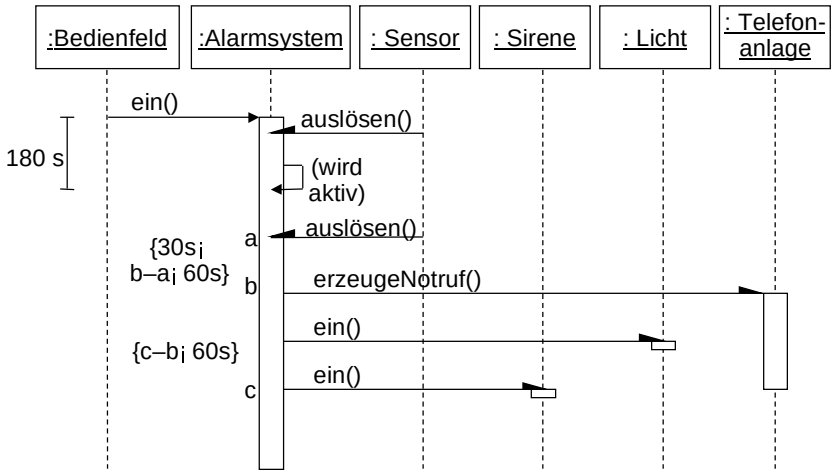
Beispiel: Innere Kapseln



Beispiel: ROOM/UML-Instanzendiagramm für Gesamtsystem



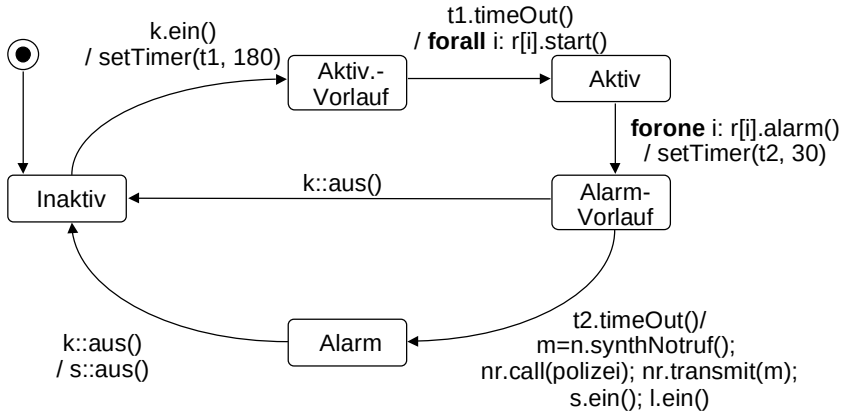
Sequenzdiagramme für Echtzeitsysteme



Zeitbedingungen schon in Anforderungsbeschreibung und -Analyse!

Zustandsdiagramme in UML/ROOM

Beispiel: Zustandsdiagramm für Kapsel "Alarmsteuerung"



Hinweis: Darstellung ist nur angelehnt an realen Aktionssprachen!

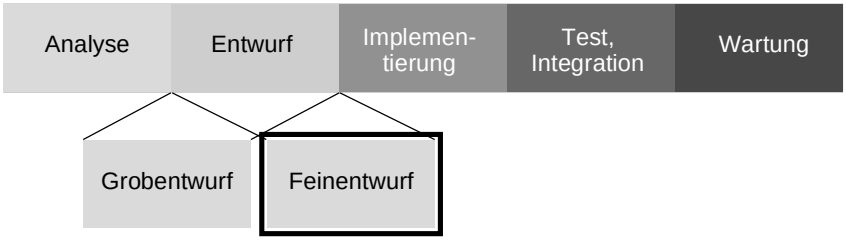
Aktionssprachen für Realzeitsysteme

- Eingebettet in UML als "Aktionen" von Zustandsdiagrammen
- Programmiersprache:
  - Sequentielle Komposition
  - Evtl. Schleifenkonstrukte
  - Berechnungen
  - Senden von Signalen über Ports
  - Aufrufen von Systemdiensten (z.B. *Timer* = Stoppuhr)
    - » ROOM/UML: Interner Port "TimeServiceSAP"
- Ablauflogik und Fallunterscheidungen in Zustandsdiagramm
- Ereignisse:
  - Eintreffen von Signalen auf Ports
  - Systemereignisse (z.B. *timeout*)
- Simulierbar und compilierbar zu Programmcode

UML/ROOM: Zusammenfassung

- Ausgereifte, in UML eingebettete Darstellungsformen
  - Vorgabe wichtiger, allerdings fixierter, Standard-Architekturelemente
  - Abgrenzung zu Standard-UML etwas unklar
  - Konkurrierender Ansatz basierend auf Rhapsody
- Sehr gute Werkzeug-Unterstützung vorhanden, Dokumentation/Methodik leider unbefriedigend
- Fliessender Übergang zwischen Analyse, Entwurf und Implementierung
  - Spezifikation = abstrakte Sicht auf Implementierung
  - Simulation von Spezifikationen
- Daten- und Verhaltenskapselung wird unterstützt
- Unterstützung für mehrfache Instanzen desselben Verhaltens
  - Prinzipiell gegeben (besser als z.B. in SA/RT)
  - Relativ kompliziert
    (warum keine Konnektoren in Klassendiagrammen?)

4.
Softwarearchitektur und Softwareentwurf
4.4
Objektorientierter Systementwurf



Literatur: Sommerville 14
Balzert LE 31
Jalote 6

Objektorientierter Komponentenentwurf

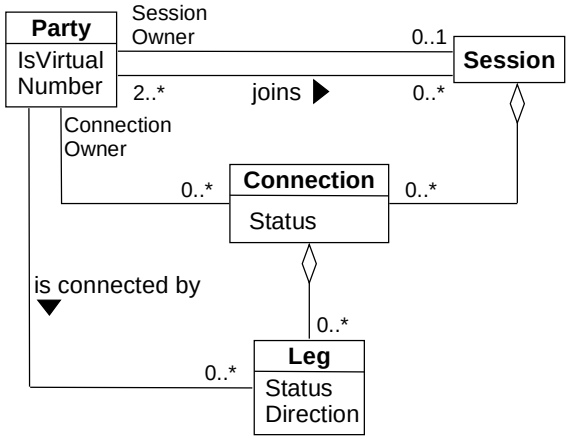
- Ausgangspunkt:
  - Grobdefinition der Architektur:
    - » Zerlegung in Subsysteme
    - » Verteilungskonzept
    - » Ablaufmodell
- Ergebnis:
  - OO-Modell für jedes Subsystem der Architektur
  - OO-Modell für unterstützende Subsysteme
  - Spezifikationen der Klassen
  - Spezifikationen von externen Schnittstellen

Verfeinerung des Analysemodells

- Entwurfsmodell ist detaillierter als Analysemodell
- Listen der Attribute und Operationen: vollständig
- Attribute und Operationen: Datentypen, Sichtbarkeit
- Operationen: Spezifikation
  - z.B. Vor- und Nachbedingungen
- Assoziationen/Aggregationen: Optimierung der Navigation
  - Navigationsrichtung, Ordnung, Qualifikation
- Einbindung in Infrastruktur, Altsysteme etc.
- Verwaltungsklassen

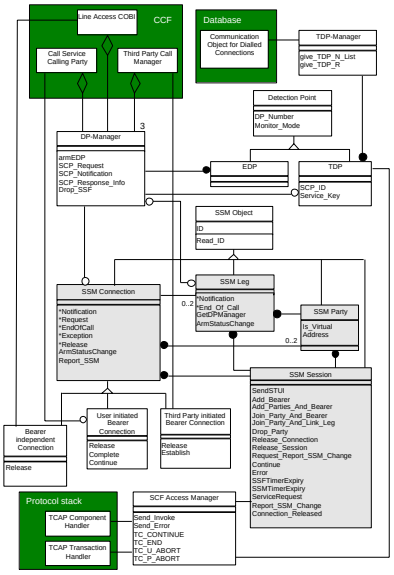
- Zusammenhang zu den Anforderungen:
  - Schlüsselszenarien als detaillierte Sequenzdiagramme

Beispiel: Analysemodell



Auszug aus einem realen Projekt im Bereich der Telekommunikation (Session-Management)

Beispiel: Entwurfsmodell (Überblick)



Notation: OMT

Teilaufgaben beim Objektorientierten Entwurf

- Verfeinerung des Analysemodells
  - OOD umfaßt mehr Detailinformation als OOA
- Vervollständigung der Architektur
  - Anbindung an Benutzungsoberfläche
  - Anbindung an Datenhaltung
  - Anpassung an gewählte Verteilung
  - Ablaufsteuerung
- Optimierung des Entwurfs
- Anpassung an Implementierungssprache

Entwurfs-muster