Technische Universität Berlin
Institut für Kommunikations- und Softwaretechnik
FG Softwaretechnik
Franklinstr. 28/29
10587 Berlin

Santen
Nordwig

Klausur

Softwaretechnik

24. Februar 2000

Prüfen Sie zunächst, ob dieses Exemplar vollständig ist (17 Blätter ohne Deckblatt).

Tragen Sie auf diesem Deckblatt und darüber hinaus auf allen Blättern, die Sie für Ihre Niederschrift verwenden, Ihren Namen und Ihre Matrikelnummer ein.

Zum Bestehen der Klausur sind **40 Punkte** notwendig. Insgesamt sind 100 Punkte möglich. Die Bearbeitungszeit beträgt 120 Minuten. In der Klausur sind **außer Telekommunikationseinrichtungen sämtliche Hilfsmittel** zugelassen.

Viel Erfolg!

Name, Vorname: ____

Matrikelnummer: ____________________

Studienrichtung/Prüfungsordnung: ____________________
SWT-Übungen habe ich im Semester ____________besucht.

Aufgabe	1	2	3	Gesamt
Maximal:	40	35	25	100
Erreicht:				

Aufgabe 1 (Fusion)

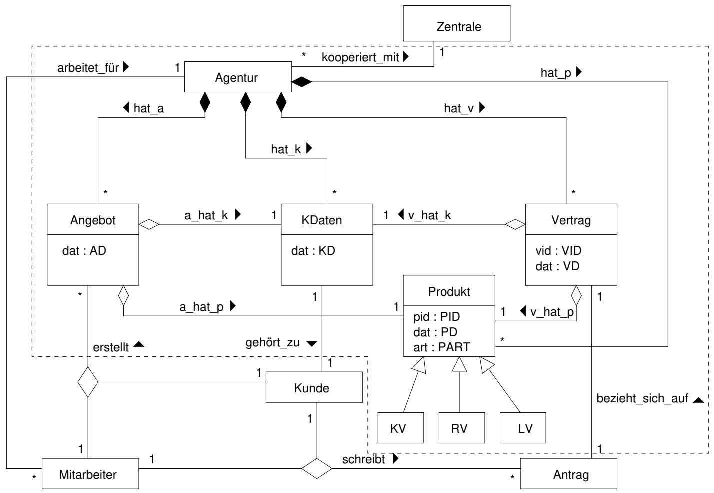
(Geschätzte Bearbeitungszeit: 50 min.)

Es soll ein Softwaresystem für eine Versicherungsagentur entwickelt werden. Diese Agentur habe unter anderem folgende Aufgaben:

- Es werden nur folgende Produkte angeboten: Kranken- (KV), Renten- (RV) und Lebensversicherungen (LV).
- Ein Mitarbeiter erstellt für die Kunden Angebote für einzelne Versicherungsarten, die die wesentlichen Daten für die jeweilige Versicherung enthalten. Die Produktinformationen müssen in Form einer Datenbank vorliegen. Die Kundendaten werden eingegeben, falls sie noch nicht vorhanden sind.
- Der Mitarbeiter füllt zusammen mit dem interessierten Kunden ein Antragsformular aus, das dann per Post an die Zentrale geschickt wird. Die Zentrale prüft den Antrag und erfasst die Daten in ihrem System. Kommt der Vertrag zustande, sendet die Zentrale anschließend diesen Vertrag elektronisch an die Agentur.
- Monatlich erfolgt ein Datenabgleich über Produktänderungen und -einführungen, indem die Zentrale die Datenbanken der Agenturen für die Produkte einzeln aktualisiert.

Die Entwicklung soll mit Hilfe der Fusion-Methode durchgeführt werden. Betrachtet werden sollen *nur* die genannten Funktionalitäten.

1. Zeichnen Sie in das gegebene Klassendiagramm die Systemgrenze ein, und nennen Sie die Akteure. *5 Punkte*
2. Erstellen Sie ein UseCase-Modell, das die o.g. Anforderungen realisiert. *5 Punkte*
3. Geben Sie zu den gefundenen UseCases Timeline-Diagramme an, die konkrete Szenarien aufzeigen. *10 Punkte*
4. Vervollständigen Sie das Operationsmodell um die identifizierten Systemoperationen. Geben Sie *nur* die Teile *Name*, *Description*, *Input*, *Reads*, *Changes* und *Sends* an. Für die im Teil 5 und 6 betrachtete Systemoperation *prod_update* brauchen Sie hier kein Operationsschema aufzustellen. *10 Punkte*
5. Geben Sie für die Systemoperation *prod_update* den Objekt-Interaktionsgraphen an. *5 Punkte*
6. Stellen Sie das Referenzierungsmodell *nur* für den Objekt-Interaktionsgraphen von *prod_update* auf. *5 Punkte*

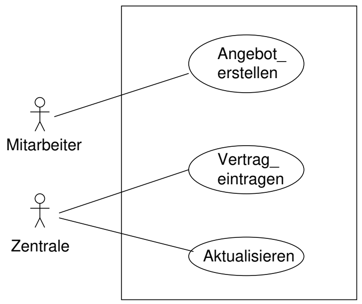


Abkürzungen:
PD - Produktdaten, AD - Angebotsdaten, KD - Kundendaten,
VD - Vertragsdaten, PART - Produktart,
VID - Vertragsnummer und PID - Produktnummer

1. Akteure:

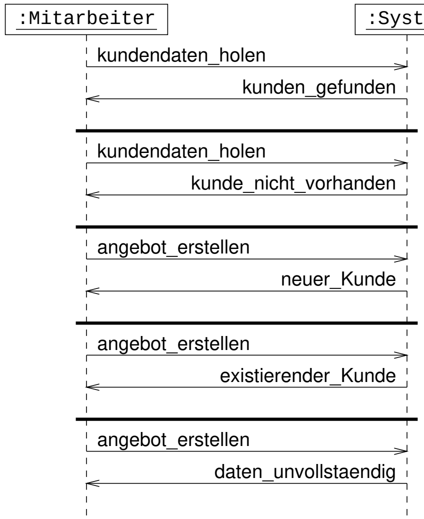
Zentrale, Mitarbeiter

2. UseCase-Modell:

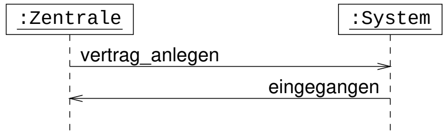


3. Timeline-Diagramme:

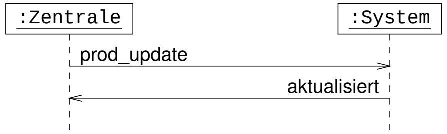
Angebot_erstellen



Vertrag_eintragen



Aktualisieren



4. Operationsmodell:

Name: prod_update
Description: Die Zentrale aktualisiert ein Produkt bei der Agentur. Ein bereits vorhandenes Produkt wird entsprechend geändert, noch nicht vorhandene werden neu angelegt.
Input: $pid : PID;\ dat : PD;\ art : PART$
Reads: $x : Agentur$
Changes: $p : Produkt\ \mathbf{new};\ hat\_p$
Sends: $Zentrale\{aktualisiert\}$

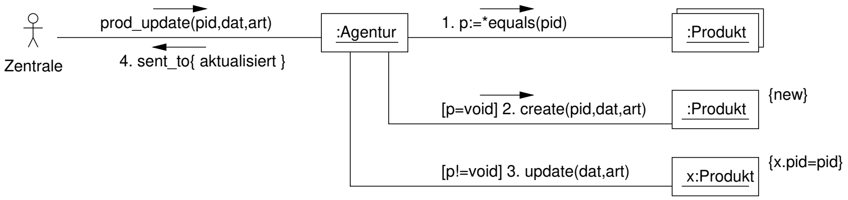
Name: kundendaten_holen

Description: Ein Mitarbeiter sucht die Kundendaten anhand eines Musters. Falls Daten gefunden werden, wird eine Menge von Kundendaten zurückgegeben.
Input: $muster : Muster$
Reads: $x : Agentur$
Sends: $Mitarbeiter\{kunden\_gefunden, kunde\_nicht\_vorhanden\}$

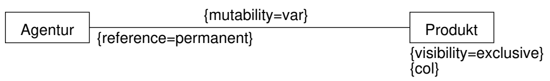
Name: angebot_erstellen
Description: Ein Mitarbeiter gibt ein Angebot für einen Kunden ein.
Input: $kd : KD;\ ad : AD;\ id : PID$
Reads: $x : Agentur$
Changes: $a : Angebot$ **new**; $k : KDaten$ **new**; $hat\_k;\ hat\_a;\ a\_hat\_k;\ a\_hat\_p$
Sends: $Mitarbeiter\{neuer\_Kunde, existierender\_Kunde, daten\_unvollstaendig\}$

Name: vertrag_anlegen
Description: Die Zentrale schickt einen Vertrag zu (online!).
Input: $kd : KD;\ pid : PID;\ dat : VD;\ vid : VID$
Reads: $x : Agentur$
Changes: $v : Vertrag$ **new**; hat_v; $k : KDaten$ **new**; $hat\_k;\ v\_hat\_k;\ v\_hat\_p$
Sends: $Zentrale\{eingegangen\}$

5. Objekt-Interaktionsgraph (prod_update):



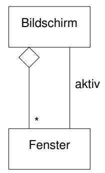
6. Referenzierungsmodell (nur prod_update):



Aufgabe 2 (Object-Z)

(Geschätzte Bearbeitungszeit: 30 min.)

Es ist ein einfaches Fenstersystem in Object-Z zu spezifizieren. Dieses System besteht aus einer Anzahl von Fenstern, die auf einem Bildschirm dargestellt werden (vgl. nebenstehendes Klassendiagramm). Die Fenster auf dem Bildschirm können über einen Mauszeiger aktiviert und bearbeitet werden.



Gegeben seien zwei globale Konstanten h_0 und w_0.

$| \; h_0, w_0 : \mathbb{N}$

Ein $Fenster$ wird durch seine Ursprungskoordinaten x und y, seine Höhe h und seine Breite w beschrieben. Der Zustands des Fensters enthält zusätzlich noch die Menge $pixels$ der von diesem Fenster belegten Pixelkoordinaten.

Initial sind die Höhe und die Breite eines Fensters durch die Konstanten h_0 und w_0 gegeben. Die Ursprungskoordinaten werden von dem Fenstersystem gemäß der weiter unten beschriebenen Einschränkungen gewählt.

Die Operation $schiebe$ verschiebt ein Fenster um die Anzahl $dx?$ von Pixeln in der Breite und die Anzahl $dy?$ von Pixeln in der Höhe. Die Eingaben $dx?$ und $dy?$ der Operation sind ganze Zahlen. Sie können also insbesondere auch negativ sein.

Ein $Bildschirm$ hat eine feste Dimension: Seine Höhe H beträgt 600 Pixel, seine Breite W beträgt 800 Pixel.

Ein Bildschirm aggregiert eine Menge von Fenstern. Diese sind auf dem Bildschirm *schnittfrei nebeneinander* angeordnet, d.h. ein Pixel des Bildschirms wird höchstens von einem Fenster belegt.

Eines oder mehrere der auf dem Bildschirm dargestellten Fenster können *aktiv* sein. Die Menge der aktiven Fenster wird im Zustand des Bildschirms verwaltet.

Initial werden auf dem Bildschirm keine Fenster dargestellt.

Ein Bildschirm bietet drei Operationen an:

- Die Operation $neues\_fenster$ erzeugt ein neues initialisiertes Fenster auf dem Bildschirm. Die Position des Fensters wird intern vom System bestimmt. Die Operation gibt eine Referenz auf dieses Fenster als Ausgabe an die aufrufende Instanz zurück.
- Die Operation $aktiviere\_fenster$ fügt der Menge der zur Zeit aktiven Fenster das Fenster hinzu, das unter dem Mauszeiger liegt. Die Koordinaten des Mauszeigers sind Eingaben dieser Operation.
- Die Operation $schiebe\_aktiv$ verschiebt *alle* aktiven Fenster des Bildschirms um *dieselbe* Anzahl von Pixeln $dx?$ horizontal und $dy?$ vertikal.

Aufgabenstellung:

Die oben beschriebene Funktionalität des Fenstersystems soll in den zwei Object-Z-Klassen $Fenster$ und $Bildschirm$ spezifiziert werden. Das Zustandsschema der Klasse $Fenster$ ist vorgegeben:

Fenster

$x, y : \mathbb{N}$
$h, w : \mathbb{N}$
$pixels : \mathbb{P}(\mathbb{N} \times \mathbb{N})$

$pixels = (x \mathrel{..} x + w - 1) \times (y \mathrel{..} y + h - 1)$

a.) Spezifizieren Sie die Initialisierung eines Fensters. *2 Punkte*

INIT

b.) Spezifizieren Sie die Verschiebeoperation $schiebe$ eines Fensters. *4 Punkte*

schiebe

Betrachten Sie nun die Klasse $Bildschirm$:

Bildschirm

c.) Spezifizieren Sie den konstanten und den variablen Teil des Zustands der Klasse $Bildschirm$. *11 Punkte* d.) Spezifizieren Sie die Initialisierung der Klasse $Bildschirm$. 1 Punkte

$INIT$

e.) Spezifizieren Sie die Operationen $neues\_fenster$ und $aktiviere\_fenster$. 12 Punkte

$neues\_fenster$

$aktiviere\_fenster$

f.) Spezifizieren Sie die Operation $schiebe\_aktiv$ unter Verwendung geeigneter Operatoren des Schemakalküls. *5 Punkte*

$schiebe\_aktiv ==$

Musterlösung

$h_0, w_0 : \mathbb{N}$

Fenster

$x, y : \mathbb{N}$
$h, w : \mathbb{N}$
$pixels : \mathbb{P}(\mathbb{N} \times \mathbb{N})$

$pixels = (x \mathinner{..} x + w - 1) \times (y \mathinner{..} y + h - 1)$

INIT

$h = h_0$
$w = w_0$

schiebe

$\Delta(x, y, pixels)$
$dx?, dy? : \mathbb{Z}$

$x' = x + dx?$
$y' = y + dy?$

Schirm

$H, W : \mathbb{N}$

$H = 600$
$W = 800$

$F : \mathbb{P}\ Fenster$
$aktiv : \mathbb{P}\ Fenster$

$\forall f, g : F \mid f \neq g \bullet f.pixels \cap g.pixels = \varnothing$
$aktiv \subseteq F$

INIT

$F = \varnothing$

neues_fenster

$\Delta(F)$
$neu! : Fenster$

$neu! \notin F$
$neu!.INIT$
$F' = F \cup \{neu!\}$

aktiviere_fenster

$\Delta(aktiv)$
$x?, y? : \mathbb{N}$

$aktiv' = aktiv \cup \{f : F \mid (x?, y?) \in f.pixels\}$

$schiebe\_aktiv == \bigwedge f : aktiv \bullet f.schiebe$

Aufgabe 3 (Statecharts)

(Geschätzte Bearbeitungszeit: 30 min.)

Ein Gebäude ist mit einer automatischen Zugangskontrolle ausgestattet. Berechtigte Personen erhalten eine Chipkarte, mit der sie das Gebäude über die einzige vorhandene Tür betreten können. An dieser Tür befindet sich ein Chipkartenleser, der die Verriegelung der Tür steuert. Eine Leuchtdiode (LED) am Kartenleser zeigt den Status der Türverriegelung an.

Die Zugangskontrolle erfüllt folgende Anforderungen:

- Die Tür besitzt eine Verriegelung, die von der Zugangskontrolle gesteuert wird. Ist die Tür verriegelt, dann lässt sie sich nicht öffnen. Ist die Tür entriegelt, dann kann sie geöffnet werden. Eine offene Tür kann nicht verriegelt werden.
- Das Öffnen und Schliessen der Tür geschieht von Hand. Ein Sensor übermittelt der Zugangskontrolle mit den Ereignissen $oeffnen$ und $schliessen$, ob die Tür geöffnet oder geschlossen wird.
- Verriegeln und Entriegeln der Tür geschieht durch die Steuerung im Kartenleser (Ereignisse $verriegeln$ und $entriegeln$).
- Die LED zeigt an, ob der Zugang in das Gebäude erlaubt wird oder nicht. Bevor eine Karte in den Kartenleser geschoben wird, ist die LED aus. Ermöglicht die Karte den Zugang zum Gebäude, zeigt die LED ein grünes Licht. Wird der Zugang zum Gebäude verweigert, dann zeigt die LED ein rotes Licht.
- Die Steuerung der Zugangskontrolle zeigt folgendes Verhalten:
  - Wird in den leeren Kartenleser eine Chipkarte eingeführt (Ereignis k_ein), werden die Daten aus der Karte ausgelesen.
  - Wird die Karte aus dem Kartenleser entfernt (Ereignis k_aus), dann wird die Tür entriegelt, falls eine Zugangsberechtigung für diese Karte vorliegt (Bedingung ber).
  - Innerhalb von 30 Sekunden nach dem Entriegeln kann die Tür geöffnet werden. Sobald sie danach wieder geschlossen wird, wird die Tür vom System wieder verriegelt. Die Tür wird auch verriegelt, wenn 30 Sekunden verstreichen, ohne dass die Tür geöffnet wurde. Wenn die Tür wieder verriegelt wird, wird die LED abgestellt.
  - Da das System eine geöffnete Tür nicht selbständig schließen kann, wird ein Alarm ausgelöst, wenn die Tür 30 Sekunden nach Entriegelung noch offen ist. In diesem Fall zeigt die LED rotes Licht.
  - Im Alarmzustand wird die Tür verriegelt sobald sie geschlossen wird.
  - Der Alarmzustand des Systems kann nicht mehr verlassen werden.

  - Wird die Chipkarte nach dem Lesen aus dem Kartenleser gezogen und es liegt *keine* Berechtigung für diese Karte vor, dann bleibt die Tür gesperrt. Außerdem wird der Kartenleser für 10 Sekunden gesperrt und an der LED rotes Licht gezeigt. Nach diesen 10 Sekunden wird die LED abgestellt, und der Kartenleser ist bereit eine neue Chipkarte aufzunehmen.

Aufgabenstellung:

25 Punkte

Modellieren Sie das oben beschriebene Zugangskontrollsystem mit den Komponenten Kartenleser, Tür und LED mit Hilfe von Statecharts.