Technische Universität Berlin
Institut für Softwaretechnik und Theoretische Informatik
FG Softwaretechnik
Franklinstr. 28/29
10587 Berlin

Jähnichen
Bittner
Herrmann
Mosconi

Klausur

Softwaretechnik

Sommersemester 04, 21. Juli 2004

Prüfen Sie zunächst, ob dieses Exemplar vollständig ist (*16* Blätter ohne Deckblatt und Leerseiten). Bitte keine eigenen Blätter verwenden.

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 **35 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: ______________________

SWT-Übungen habe ich im Semester ____________ besucht.

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

Aufgabe 1 (Fusion)

(Geschätzte Bearbeitungszeit: 40 min.)

Ein Tippsystem für die Fussballweltmeisterschaft 2006

Für die Fussballweltmeisterschaft 2006 möchte die Tippgemeinschaft SWT-Tipp eV. ein System entwickeln, das den Anforderungen der Tippgemeinschaft genügt. Der Verein muss aus mindestens 4 Mitgliedern bestehen. Mit Hilfe des Systems soll es möglich sein

- neue Mitglieder in die Tippgemeinschaft aufzunehmen
- Tipp-Ergebnisse auszuwerten
- Wetteinsätze und Mitglieder zu verwalten

Jedes *Mitglied* der Wettgemeinschaft zahlt einmalig bei Aufnahme einen festen Betrag an den Administrator. Jedes Mitglied kann maximal so viele Tipps abgeben, wie Spiele stattfinden. Ein Tipp kann spätestens eine halbe Stunde vor Anpfiff eines Spieles abgegeben werden.
Ein *neues Mitglied* meldet sich über die Website im System an. Dabei wird ein neuer Bereich für dieses Mitglied angelegt und das vom Neumitglied gewählte Passwort im System aufgenommen. Ebenso wird sein Wetteinsatz notiert und wird in eine Mailingliste aufgenommen. Ein *Mitglied* identifiziert sich mit Namen und Passwort im System.

Der *Administrator* stellt die bereits feststehenden Tippmöglichkeiten (Paarungen von Fußballmannschaften) der Spiele im System zur Verfügung. Der Administrator verwaltet die Mitgliederbereiche. Er ist für Zahlung der Wetteinsätze und am Ende für die Auszahlung der Siegerprämien zuständig.

Eine bekannte Zeitung *Kicker* trägt die Spielergebnisse in das System ein. Die Auswertung der Tipps kann von jedem Mitglied über die Website erfragt werden.
Für ein Tippergebnis kann es maximal 3 Punkte geben:
0 Punkte gibt es für ein fehlenden oder falschen Tipp

1 Punkt gibt es für die richtige Tendenz, d.h. Mannschaft1 gewinnt oder Mannschaft2 gewinnt oder beide spielen unentschieden, aber ein nicht korrektes Torverhältnis

3 Punkte werden für exaktes Torverhältnis vergeben

Aufgabenstellung:

Das System soll nach der Fusion-Methode entwickelt werden. Dabei soll das Verhalten der Systemoperation *aufnehmen_neues_Mitglied* genau untersucht werden.

a.) Ergänzen Sie im vorgegebenen Klassenmodell die Multiplizitäten. *4 Punkte*

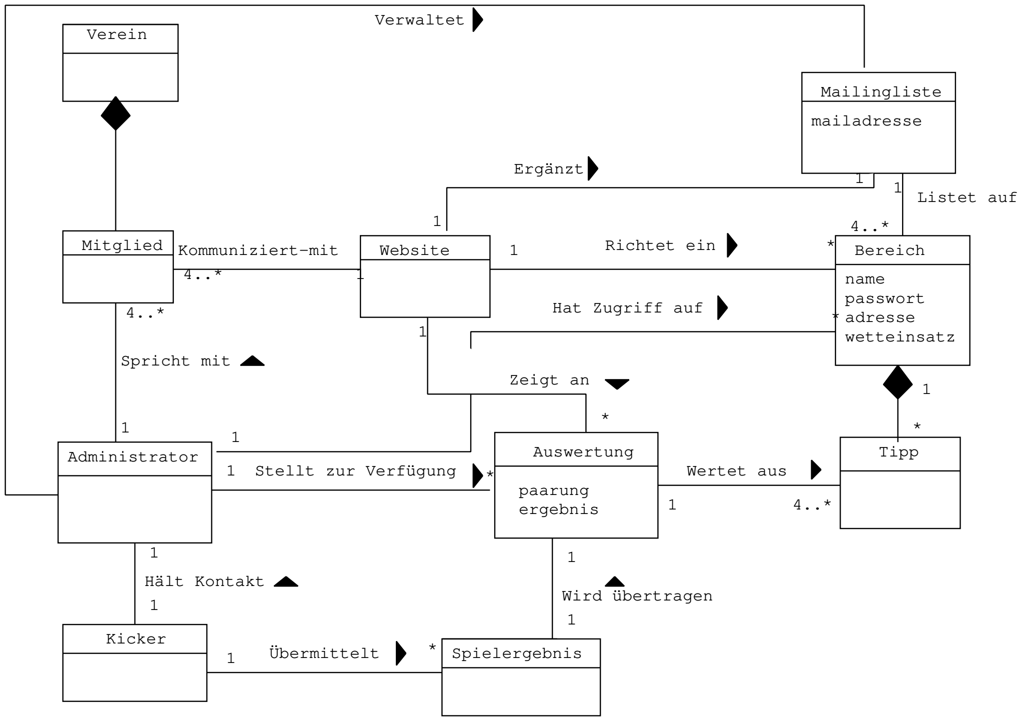
b.) Vervollständigen Sie das UseCases Modell, indem Sie die Akteure des Systems bestimmen und ihnen Uses Cases zuordnen. *3 Punkte*

c.) Vervollständigen Sie das dazugehörige Systemklassenmodell, indem Sie die Systemgrenze festlegen und ggf. weitere neue Klassen und Assoziationen einfügen. *3 Punkte*

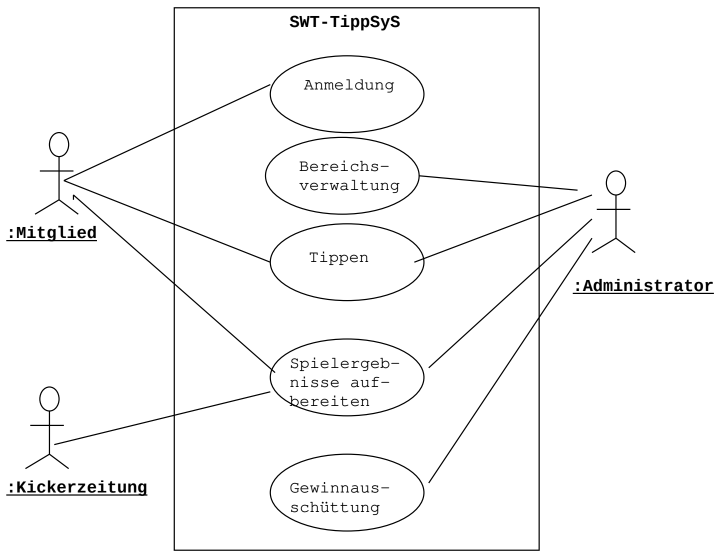
d.) Vervollständigen Sie das vorgegebene Operationsschema für die Systemoperation *aufnehmen_neues_Mitglied*. *15 Punkte*

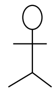
e.) Geben Sie den Objektinteraktionsgraphen für die Systemoperation *aufnehmen_neues_Mitglied* an. *10 Punkte*

Klassenmodell

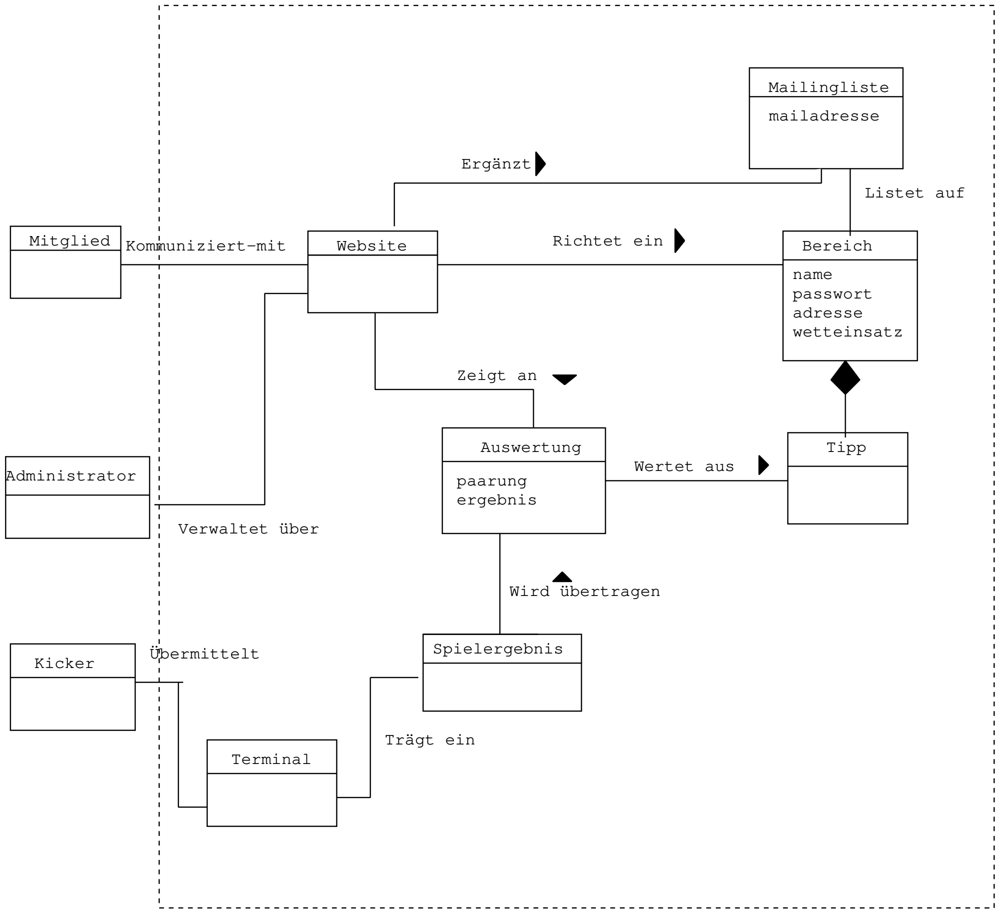


UseCase-Modell

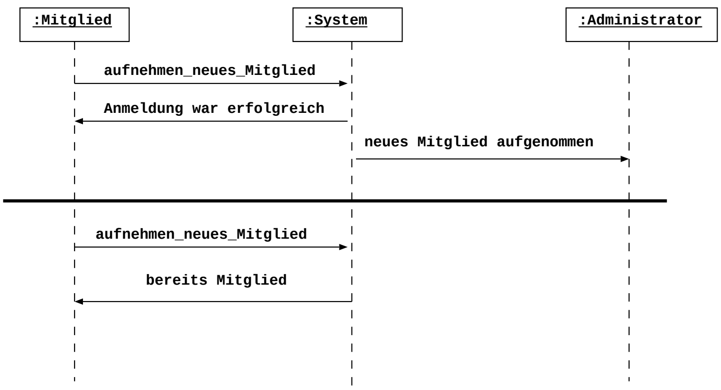




Systemklassenmodell

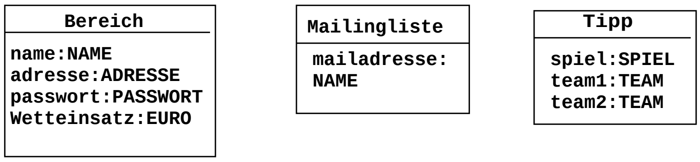


Time-LineModell (Vorgabe)



Hinweis :

Nehmen Sie die folgenden Attribut- und Typdeklarationen an:



Elementare Datentypen:

NAME	Name des Mitgliedes
ADRESSE	Adresse des Mitgliedes
PASSWORT	Zur Authentisierung
SPIEL	Fussballspiel
TEAM	Nationalteam

Operationsmodell

Operation = aufnehmen_neues_Mitglied

Description = Ein neues Mitglied will in die Tippgemeinschaft SWT-Tipp eV Fussball aufgenommen werden. Ist es bereits Mitglied des Vereins, wird eine entsprechende Fehlermeldung zurückgegeben. Ansonsten wird das neue Mitglied als auch der Administrator benachrichtigt.

Input = $name, adresse, passwort, einsatz$

Reads = $w : Website, m : Mailingliste\ with\ (w, m) \in Ergaenzt$

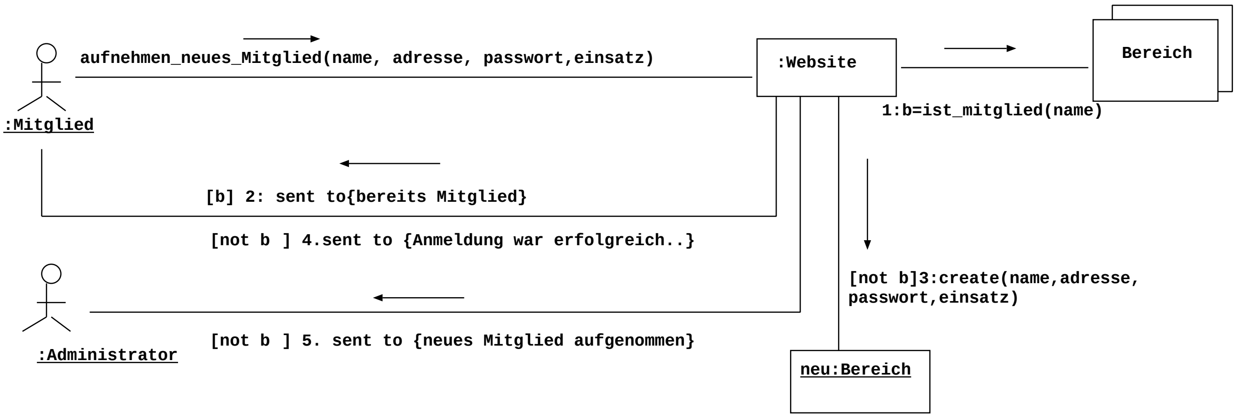
Changes = $: Bereich, neues\_mitglied : Bereichtype, Richtet\_ein, Listet\_auf$

Sends = $Mitglied\{Anmeldung\_war\_erfolgreich, ist\_bereits\_Mitglied\}$
$Administrator\{neues\_Mitglied\_ist\_aufgenommen\}$

Pre = $-$

Post = $\textbf{let}\ existiert == \{m : Bereich \mid m.name = name$
$\wedge (w, m) \in Richtet\_ein\} \bullet$
$((existiert \neq \varnothing \Rightarrow$
$\textbf{is\_sent}\ \{bereits\_Mitglied\} \wedge \textbf{no\_effect}\ ) \wedge$
$(existiert = \varnothing \Rightarrow$
$neues\_mitgliednew = \wedge$
$neues\_mitglied.name' = name \wedge$
$neues\_mitglied.adresse' = adresse \wedge$
$neues\_mitglied.passwort' = passwort \wedge$
$neues\_mitglied.wetteinsatz' = einsatz \wedge$
$Richtet\_ein' = Richter\_ein \cup \{(w, neues\_mitglied)\} \wedge$
$Listet\_auf' = Listet\_auf \cup \{neues\_mitglied, m)\} \wedge$
$\textbf{is\_sent}\ \{Anmeldung\_war\_erfolgreich\} \wedge$
$\textbf{is\_sent}\ \{neues\_Mitglied\_aufgenommen\}))$

Objektinteraktionsmodell



Aufgabe 2 (Object-Z) - Internetversand

Ein Internetversand erlaubt die online-Bestellung von Artikeln. Um Versandkosten zu sparen wird eine Bestellung erst ausgeliefert, wenn alle bestellten Artikel verfügbar sind. Bei nicht auslieferbaren Bestellungen sollen die bereits verfügbaren Artikel für den Kunden vorgemerkt werden, damit diese nicht an einen anderen Kunden ausgeliefert werden.

(Geschätzte Bearbeitungszeit: 50 min.)

Aufgabenstellung: Spezifizieren Sie das Bestellsystem in Object-Z, indem Sie die vorgegebenen Klassendefinitionen ergänzen.

Folgende globale Definitionen sind vorgegeben:

$[ARTIKEL]$
$[KUNDE]$
$Nachricht ::= vormerken\_erfolgreich \mid ausliefern\_erfolgreich \mid nicht\_alle\_verfügbar$

Lagerposten

15 Punkte

Ein Lagerposten gibt an, wieviele Exemplare eines bestimmten Artikels im Lager vorhanden sind. Im Rahmen der verfügbaren Anzahl können Artikel für einen Kunden vorgemerkt werden. Vorgemerkte Artikel stehen für andere Kunden nicht mehr zur Verfügung. Die vorgemerkte Anzahl von Exemplaren kann an den betreffenden Kunden ausgeliefert werden.

Das vorgegebene Attribut *vormerkungen* erlaubt es, für jeden Kunden eine Anzahl von vorgemerkten Exemplaren einzutragen. Kunden ohne Vormerkung des gegebenen Artikels erscheinen in dieser Abbildung nicht. Die Gesamtanzahl der vorgemerkten Exemplare des gegebenen Artikels soll redundant im Attribut *anzahl_vorgemerkt* erfaßt werden.

Für die Beziehungen zwischen Attributen der Klasse Lagerposten soll eine Invariante definiert werden.

Die Operation *Einlagern* modelliert das Auffüllen des Lagerbestandes. Die Operation *Vormerken* reserviert eine Anzahl von Exemplaren eines Artikels für einen Kunden. Es können nur vorhandene und verfügbare Exemplare vorgemerkt werden. Diese Operation darf nur für Kunden angewendet werden, die für den gegebenen Artikel noch keine Vormerkung haben. Die Operation *Ausliefern* modelliert die tatsächliche Auslieferung der vorgemerkten

Stückzahl an einen Kunden, bei der alle relevanten Anzahlen angepaßt werden müssen. Insbesondere sollen die Vormerkungen des Kunden zu dem Lagerposten gelöscht werden.

Die Operation *Vormerken* soll durch eine Nachricht signalisieren, ob sie erfolgreich war. Die Operationen *Ausliefern* soll **nicht** robust spezifiziert werden, d.h. sie hat eine Vorbedingung ungleich "true".

Lagerposten

$\upharpoonright (artikel, INIT, Einlagern, Vormerken, Ausliefern)$

$artikel : ARTIKEL$
$anzahl\_vorhanden : \mathbb{N}$
$vormerkungen : KUNDE \nrightarrow \mathbb{N}$
$anzahl\_vorgemerkt : \mathbb{N}$

$anzahl\_vorgemerkt \leq anzahl\_vorhanden$

INIT

$anzahl\_vorhanden = 0$
$vormerkungen = \varnothing$

... (Fortsetzung nächste Seite)

Lagerposten

... (Fortsetzung)

Einlagern

$\Delta(anzahl\_vorhanden)$
$anzahl? : \mathbb{N}_1$

$anzahl\_vorhanden' = anzahl\_vorhanden + anzahl?$

Vormerken_OK

$\Delta(vormerkungen, anzahl\_vorgemerkt)$
$kunde? : KUNDE$
$anzahl? : \mathbb{N}$
$n! : Nachricht$

$kunde? \notin \mathrm{dom}\, vormerkungen$
$anzahl? \leq anzahl\_vorhanden - anzahl\_vorgemerkt$
$vormerkungen' = vormerkungen \cup \{kunde? \mapsto anzahl?\}$
$anzahl\_vorgemerkt' = anzahl\_vorgemerkt + anzahl? \land$
$n! = vormerken\_erfolgreich$

Vormerken_nicht_verfügbar

$kunde? : KUNDE$
$anzahl? : \mathbb{N}$
$n! : Nachricht$

$anzahl > anzahl\_vorhanden - anzahl\_vorgemerkt$
$n! = nicht\_alle\_verfügbar$

$Vormerken \mathrel{\widehat{=}} Vormerken\_OK \,[]\, Vormerken\_nicht\_verfügbar$

Ausliefern

$\Delta(anzahl\_vorhanden, vormerkungen, anzahl\_vorgemerkt)$
$kunde? : KUNDE$
$n! : Nachricht$

$\textbf{let}\ anzahl == vormerkungen(kunde?) \bullet$
$\quad anzahl\_vorhanden' = anzahl\_vorhanden - anzahl \land$
$\quad vormerkungen' = vormerkungen \lhd \{kunde?\} \land$
$\quad anzahl\_vorgemerkt' = anzahl\_vorgemerkt - anzahl \land$
$\quad n! = ausliefern\_erfolgreich$

Bestellposten

12 Punkte

Wenn ein Kunde eine Anzahl Exemplare eines Lagerpostens bestellt, wird dazu ein Bestellposten angelegt. Welche Zustände ein Bestellposten durchlaufen kann, wird durch den Datentyp *Zustand* modelliert. Die Hilfsoperation *Setze_zustand* soll für die beiden Operationen *Vormerken* und *Ausliefern* bereitgestellt werden. Diese beiden Operationen sollen argumentlos definiert werden und die nötigen Änderungen am Lagerposten veranlassen. *Ausliefern* soll nur anwendbar sein, wenn ein vorheriges *Vormerken* erfolgreich war.

Hinweis: Für die Klasse Bestellposten sollen 4 Zustandsvariablen angegeben werden.

Bestellposten

$\upharpoonright (zustand, kunde, INIT, Definieren, Vormerken, Ausliefern)$

$Zustand ::= leer \mid definiert \mid lieferbar \mid ausgeliefert$

$kunde : KUNDE$
$lagerposten : Lagerposten$
$anzahl : \mathbb{N}$
$zustand : Zustand$

INIT

$zustand = leer$

Definieren

$\Delta(kunde, lagerposten, anzahl, zustand)$
$kunde? : KUNDE$
$artikel? : ARTIKEL$
$anzahl? : \mathbb{N}$

$zustand = leer$
$kunde' = kunde?$
$lagerposten'.artikel = artikel?$
$anzahl' = anzahl?$
$zustand' = definiert$

Setze_zustand

$\Delta(zustand)$
$n? : Nachricht$

$n? = vormerken\_erfolgreich \Rightarrow zustand' = lieferbar$
$n? = liefern\_erfolgreich \Rightarrow zustand' = ausgeliefert$

$Vormerken \mathrel{\widehat{=}}$
$\quad lagerposten.Vormerken[kunde/kunde?, anzahl/anzahl?];$
$\quad Setze\_zustand$

$Ausliefern \mathrel{\widehat{=}}$
$\quad [\mid zustand = lieferbar];$
$\quad lagerposten.Ausliefern[kunde/kunde?];$
$\quad Setze\_zustand$

Bestellung

13 Punkte

Eine Menge von Bestellposten desselben Kunden wird zu einer Bestellung zusammengefaßt. Eine Bestellung ist nur auslieferbar, wenn alle ihre Bestellposten ausgeliefert werden können. Beim *Erstellen* einer Bestellung wird nur der Kunde eingetragen. Das Hinzufügen von Artikeln zur Bestellung muß in zwei Schritten spezifiziert werden:

1. Erzeugen und Eintragen eines neuen Bestellpostens
2. Definieren des Bestellpostens mit dem gewünschten ARTIKEL in der gewünschten Anzahl

Die Operationen *Ausliefern_OK* und *Nicht_auslieferbar* modellieren die beiden möglichen Fälle bei versuchter Auslieferung. Sie definieren nur die Vorbedingungen und gegebenenfalls Änderungen lokal innerhalb der Bestellung. Sie müssen mit der Veränderung der Bestellposten zu einer robusten Gesamtoperation *Ausliefern* kombiniert werden.

Es soll eine Invariante bezüglich des Kunden der Bestellung spezifiziert werden.

Bestellung

$\upharpoonright (INIT, Erstellen, Artikel\_hinzufügen, Prüfen, Ausliefern)$

$$kunde : KUNDE$$
$$bestellposten : \mathbb{F}\, Bestellposten$$
$$ausgeliefert : \mathbb{B}$$

$$\forall p : bestellposten \bullet p.kunde = kunde$$

INIT

$$bestellposten = \varnothing$$
$$ausgeliefert = \mathrm{false}$$

Erstellen

$$\Delta(kunde)$$
$$kunde? : KUNDE$$

$$kunde' = kunde?$$

Neuer_bestellposten

$$\Delta(bestellposten)$$
$$artikel? : ARTIKEL$$
$$anzahl? : \mathbb{N}$$
$$neuer\_posten! : Bestellposten$$

$$neuer\_posten!.INIT$$
$$neuer\_posten! \notin bestellposten$$
$$bestellposten' = bestellposten \cup \{neuer\_posten!\}$$

$$Artikel\_hinzufügen \mathrel{\widehat{=}} Neuer\_bestellposten; [neuer\_posten? : Bestellposten] \bullet neuer\_posten?.Definieren[kunde/kunde?]$$

Ausliefern_OK

$$\Delta(ausgeliefert)$$
$$n! : Nachricht$$

$$\forall b : bestellposten \bullet b.zustand = lieferbar$$
$$ausgeliefert' = \mathrm{true}$$
$$n! = ausliefern\_erfolgreich$$

Nicht_auslieferbar

$$n! : Nachricht$$

$$\exists b : bestellposten \bullet b.zustand \neq lieferbar$$
$$n! = nicht\_alle\_verfügbar$$

$$Ausliefern \mathrel{\widehat{=}} ((\bigwedge b : bestellposten \bullet b.Ausliefern);\ Ausliefern\_OK)\ [\,]\, Nicht\_auslieferbar$$

Aufgabe 3 (Statecharts) - Fahrkartenautomat

(Geschätzte Bearbeitungszeit: 30 min.)

Modellieren Sie den im Folgenden beschriebenen Fahrkartenautomaten mit Hilfe von Statecharts.
Der Fahrkartenautomat bestehe aus den folgenden Komponenten:

- Display
- Display-Beleuchtung
- Münzeinwurf
- Ausgabefach (Beleuchtung)
- Steuerung

Auswahl:
Der betrachtete Fahrkartenautomat dient dem Verkauf von Bahn-Fahrkarten. Zur Auswahl stehen 2 Tarife (1 Euro oder 2 Euro pro Fahrkarte), für die es jeweils eine Taste gibt, mit der eine Karte des entsprechenden Tarifs ausgewählt werden kann. Es können beliebig viele Karten beider Tarife durch wiederholtes Drücken der beiden Tasten ausgewählt werden. Der Automat summiert die Einzelpreise der Karten und zeigt den zu zahlenden Gesamtbetrag im Display an.

Bezahlung und Ausgabe:
Hat der Benutzer die gewünschte Anzahl Karten gewählt, kann er mit der 'OK'-Taste die Bezahlung einleiten. Bezahlt wird ausschließlich mit 1 Euro-Münzen. Dafür öffnet sich der Münzeinwurf und der Benutzer muss so viele 1 Euro-Münzen einwerfen, bis der Gesamtbetrag erreicht ist. Hat er genug Münzen eingeworfen, schließt sich der Münzeinwurf wieder und die Fahrkarten werden im Ausgabefach ausgegeben. Das Ausgabefach wird dabei für 5 Sekunden beleuchtet. Anschließend geht der Automat in den Ruhezustand über (s.u.).
Hinweis: Der zu zahlende und der bereits eingeworfene Betrag sollen als Variablen modelliert werden.

Ruhezustand:
Der Fahrkartenautomat befindet sich initial in einem Ruhezustand, in dem

das Display nicht beleuchtet ist und die Uhrzeit anzeigt. Außerdem ist der Münzeinwurf geschlossen und das Ausgabefach nicht beleuchtet. Wird eine der Tasten betätigt, geht die Beleuchtung des Displays an und statt der Uhrzeit wird der zu zahlende Betrag angezeigt. Der Benutzer kann nun wie oben beschrieben Fahrkarten kaufen.
Wird am Fahrkartenautomat für 20 Sekunden keine Taste gedrückt oder Geld eingeworfen, so geht er wieder in den Ruhezustand über, wobei alle bisherigen Eingaben verworfen werden. Bereits eingeworfene Münzen werden wieder ausgegeben (nur über die entsprechende Variable zu modellieren).

Aufgabenstellung

25 Punkte

Modellieren Sie die angegebenen Komponenten des Fahrkartenautomaten und deren Steuerung mit Hilfe von Statecharts.
Wenn Sie selber interne Ereignisse definieren, dokumentieren Sie diese *kurz* in einer Tabelle.

Verwenden Sie folgende externe Ereignisse, die durch den Benutzer ausgelöst werden können:

- t1, t2: Ticket für 1 bzw. 2 EUR gewählt
- ok: Taste 'OK' gedrückt
- geld: Einwurf einer 1 Euro-Münze