TU Berlin
Fakultät IV
Institut für Softwaretechnik und Theoretische Informatik

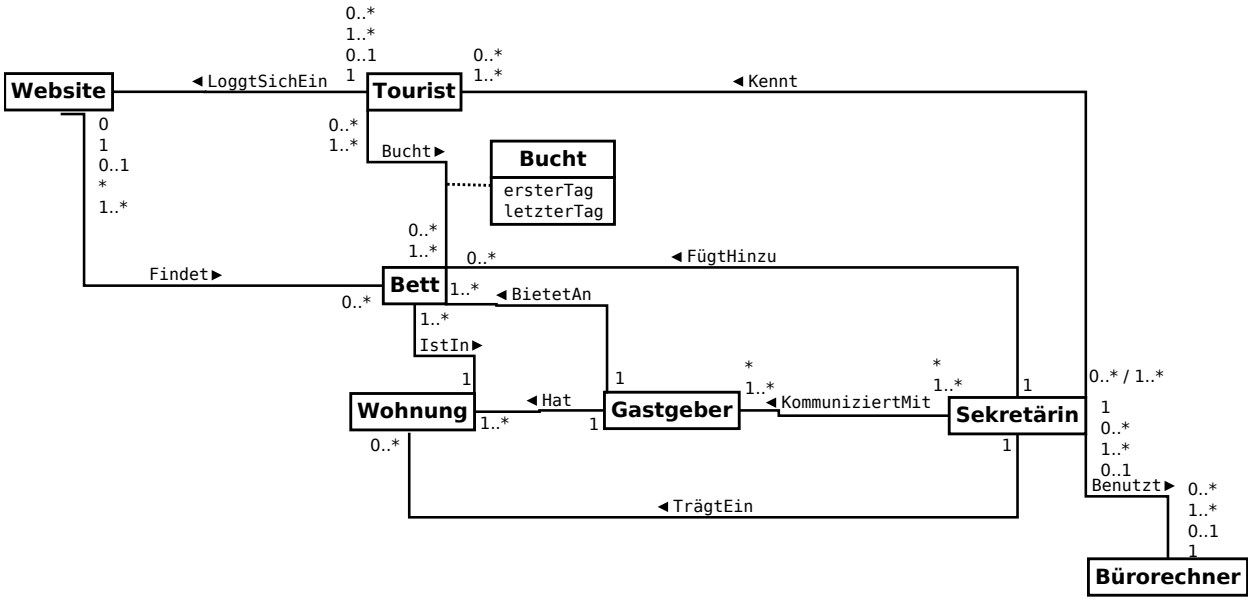
Beispiellösung der Probeklausur WS10 / 11

19. Februar 2011

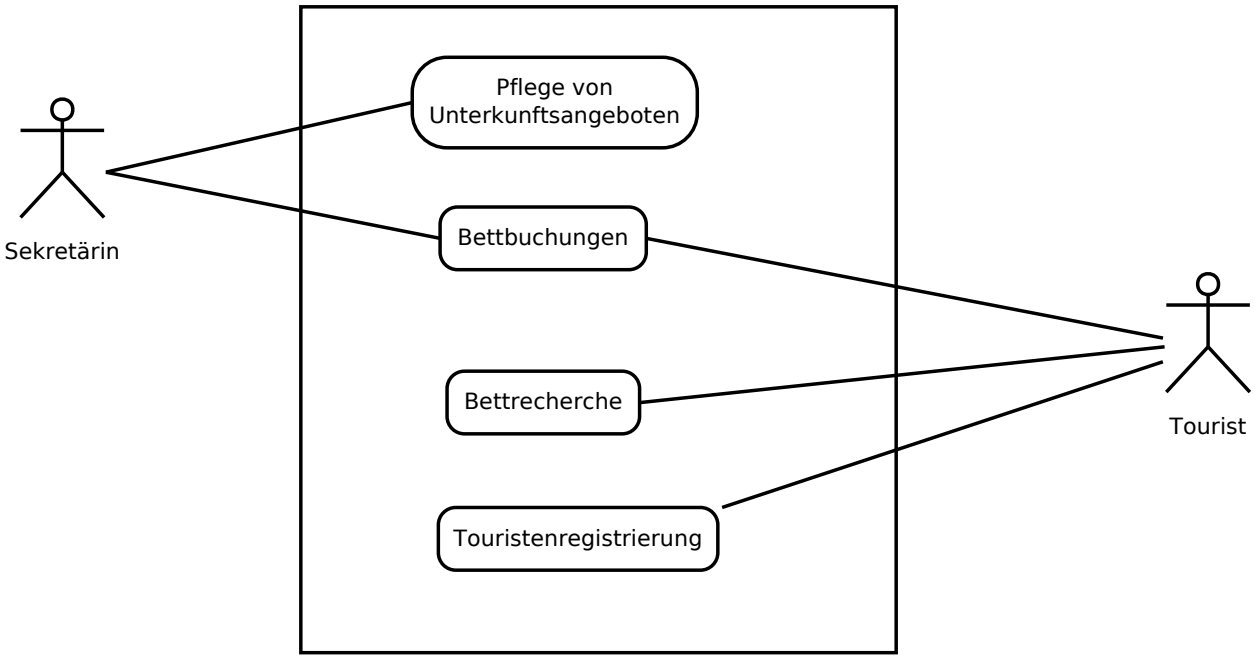
Teil I

Teil – A
Analyse & Entwurf

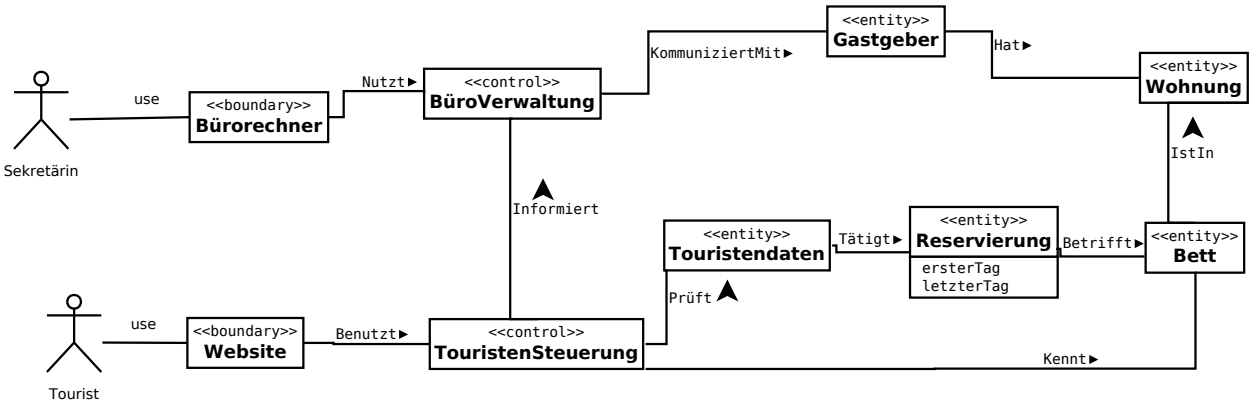
1. Klassenmodell des Gegenstandsbereichs



2. Use-Case-Diagramm



3. Systemklassenmodell / Entwurfsklassenmodell

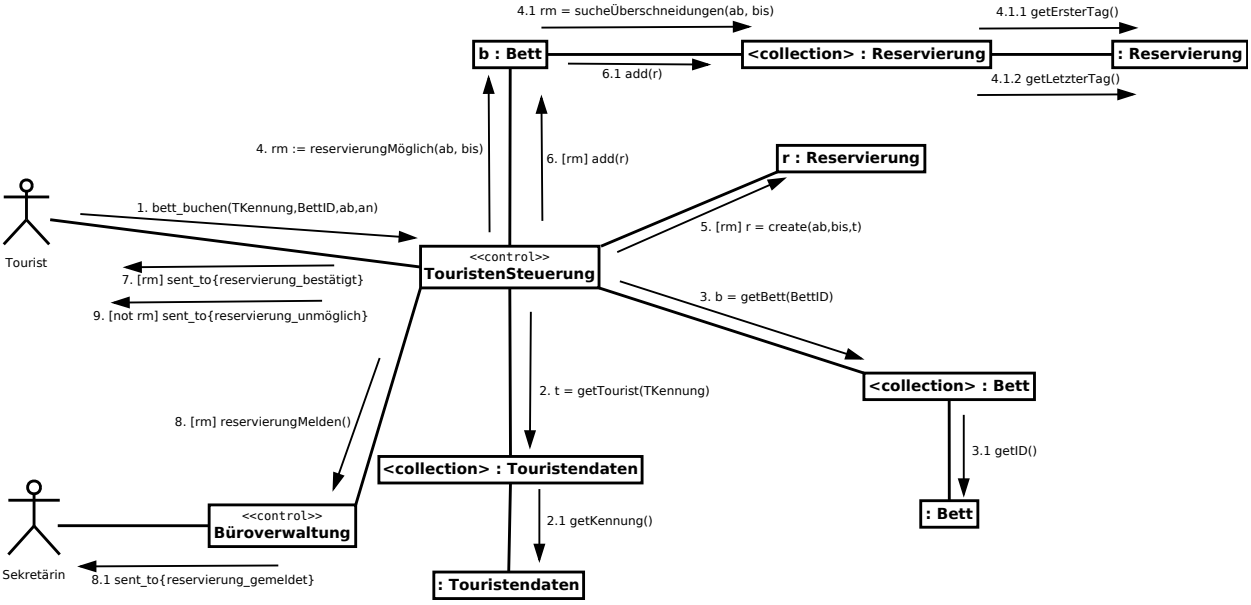


4. Operationsschema bett_buchen

Op.	=	bett_buchen
Desc.	=	Ein Tourist bucht ein bestimmtes Bett für einen angegebenen Zeitraum. **Vorbedingungen:** Es wird davon ausgegangen, dass der Tourist mit der Kennung *TKennung* im System bekannt ist und auf der Webseite eingeloggt ist. Die Existenz eines Bettes mit der Identifikation *BettID* wird vorausgesetzt.
Input	=	TKennung : String, BettID : int, ab : Date, bis : Date
Reads	=	ts : TouristenSteuerung, Kennt, Bett, Touristendaten,Prüft b : Bett **with** b.id = BettID $\wedge$ (b,ts) $\in$ Kennt t : Touristendaten **with** t.kennung = TKennung $\wedge$ (ts,t) $\in$ Prüft $\wedge$ t.eingeloggt
Changes	=	Reservierung,Tätigt, Betrifft,r: Reservierung **type**
Sends	=	Tourist{buchung_bestätigt, buchung_unmöglich}, Sekretärin{buchung_gemeldet}
Pre	=	*implicit*

Post	=	**let**
		bettreservierung == {r: Reservierung | (r,b) : Betrifft}
		überscheidene_buchungen ==
		{r:bettreservierungen | (ab $\leq$ r.ersterTag $\leq$ bis) $\vee$ (ab $\leq$ r.letzterTag $\leq$ bis)}
		$\vee$ (r.ersterTag $\leq$ ab $\wedge$ bis $\leq$ r.letzterTag);
		buchung_möglich == #überscheidene_buchungen = 0 $\wedge$ ab $\leq$ bis;
		buchung_nicht_möglich == $\neg$ buchung_möglich;
		•
		(buchung_nicht_möglich $\Rightarrow$
		is_sent{buchung_unmöglich $\wedge$ **no_effect**) $\wedge$
		(buchung_möglich $\Rightarrow$
		is_sent{buchung_bestätigt $\wedge$ **no_effect**) $\wedge$
		is_sent{buchung_gemeldet $\wedge$ **r_new**) $\wedge$
		r.ersterTag' = ab $\wedge$ r.letzterTag' = bis $\wedge$
		Tätigt' = Tätigt $\cup$ (t,r) $\wedge$ Betriff' = Betrifft $\cup$ (r,b))

5. Kommunikationsdiagramm bett buchen



6. Implementierungsklassenmodell

Bett
-id: int -Reservierung: <*> reservierungen
+getID(): int +addReservierung(r:Reservierung): void +reservierungMöglich(anfang:Datum,ende:Datum): boolean -sucheÜberscheidungen(anfang:Datum,ende:Datum)

BettCollection
-Bett: <*> betten
+getBett(bid:int): Bett

Touristendaten
-kennung: String -eingeloggt: boolean
+getKennung(): Sting

Reservierung
-ersterTag: Date -letzterTag: Date -t: Tourist
+create(anfang:Date,ende:Date,tou:Touristendaten) +getErsterTag(): Date +getLetzterTag(): Date

TouristenSteuerung
-touristen: <*>Touristendaten -betten: BettCollection -bv: Büroverwaltung
+bettBuchen(name:String,b:Bett,anfang:Date,ende:Date): void +getTourist(name:String): Tourist

Büroverwaltung
+reservierungMelden(): void

Teil II

Teil – B
Object-Z

Globale Definitionen

$Nachricht ::= Karte\_ausgelegt \mid keine\_Karte\_ausgelegt \mid Karte\_eingeworfen$
$Farbe ::= rot \mid gruen \mid blau \mid gelb$
$Zahl ::= 1 \ldots 10$

Karte

$farbe : Farbe$
$zahl : Zahl$

a) Spieler

Spieler

$\upharpoonright (INIT, hand, Karte\_einwerfen, Spielzug\_durchfuehren)$

$hand : \mathbb{F}\, Karte$

INIT

$\#hand = 8$

Karte_auslegen

$\Delta(hand)$
$liegt? : Karte$
$auslegen! : Karte$
$spieler! : Spieler$
$n! : Nachricht$

$\#hand > 0$ redundant
$\exists x : hand \mid x.farbe = liegt?.farbe \lor x.zahl = liegt?.zahl \bullet auslegen! = x$
$hand' = hand \setminus \{auslegen!\}$
$spieler! = self$
$n! = Karte\_ausgelegt$

Karte_auslegen_nicht_moeglich

$liegt? : Karte$
$auslegen! : Karte$
$spieler! : Spieler$
$n! : Nachricht$

$\#hand > 0$ redundant
$\forall x : hand \bullet x.farbe \neq liegt?.farbe \land x.zahl \neq liegt?.zahl$
$auslegen! = liegt?$
$spieler! = self$
$n! = keine\_Karte\_ausgelegt$

Karte_ziehen

$\Delta(hand)$
$liegt? : Karte$

$\#hand > 0$ redundant
$\forall x : hand \bullet x.farbe \neq liegt?.farbe \land x.zahl \neq liegt?.zahl$
$\exists k : Karte \mid k \notin hand \bullet hand' = hand \cup \{k\}$
$(alternativ : \#hand' = \#hand + 1 \land hand \subset hand')$

$Spielzug\_durchfuehren \mathrel{\widehat=} Karte\_auslegen \,[]\, \left(Karte\_ziehen \fatsemi (Karte\_auslegen\_nicht\_moeglich \,[]\, Karte\_auslegen)\right)$

Karte_einwerfen

$\Delta(hand)$
$liegt? : Karte$
$auslegen! : Karte$
$spieler! : Spieler$
$n! : Nachricht$

$\#hand > 0$ redundant
$\exists x : hand \mid x.farbe = liegt?.farbe \land x.zahl = liegt?.zahl \bullet auslegen! = x$
$hand' = hand \setminus \{auslegen!\}$
$spieler! = self$
$n! = Karte\_eingeworfen$

b) Uno-Spiel

UNO_Spiel

$reihenfolge : \mathbb{N} \rightarrowtail Spieler$
$liegt : Karte$
$dran : dom\ reihenfolge$
$beendet : \mathbb{B}$

$\forall n : \mathbb{N} \bullet n < \#reihenfolge \Rightarrow n \in dom\ reihenfolge$
$(alternativ : \#reihenfolge = max(dom\ reihenfolge) + 1$
$bzw.\ dom\ reihenfolge = 0 \ldots \#reihenfolge - 1)$

INIT

$liegt.farbe = rot$
$\forall sp : ran\ reihenfolge \bullet sp.INIT$
$dran = 0$
$\neg\ beendet$

liegende_Karte

$liegt! : Karte$

$\neg\ beendet$
$liegt! = liegt$

Karte_ersetzen

$\Delta(liegt)$
$auslegen? : Karte$

$\neg\ beendet$
$liegt' = auslegen?$

naechster_Spieler

$\Delta(dran)$
$spieler? : ran\ reihenfolge$

$\#spieler?.hand > 0$
$\neg\ beendet$
$\exists n : \mathbb{N} \bullet ($
$\quad reihenfolge\ n = spieler? \land$
$\quad n = \#spieler - 1 \Rightarrow dran' = 0 \land$
$\quad n < \#spieler - 1 \Rightarrow dran' = n + 1)$

Spiel_beenden

$\Delta(beendet)$
$spieler? : Spieler$

$\neg\ beendet$
$\#spieler?.hand = 0$
$beendet'$

$Spielzug\_abeschliessen \mathrel{\widehat=} Karte\_ersetzen \fatsemi (naechster\_Spieler \Box Spiel\_beenden)$

$$Spielzug \mathrel{\widehat=} \Big(liegende\_Karte \parallel \Big(reihenfolge(dran).Spielzug\_durchfuehren \ [alternativ :\Box\ sp : Spieler \mid reihenfolge(dran) = sp \bullet sp.Spielzug\_durchfuehren] \ \Box\ (\Box\ sp : ran\ reihenfolge \bullet sp.Karte\_einwerfen)\Big)\Big) \fatsemi Spielzug\_abschliessen$$

2) Statechart