# Software Engineering I

Lösungsvorschläge zur Klausur vom 11.8.2007

## Aufgabe 1

Gefordert war ein redundanzfreies Klassendiagramm für die beschriebene Anwendungsdomäne. Zwei (von verschiedenen möglichen) Lösungen sind in den Abbildungen 1 und 2 dargestellt. Der erste Lösungsvorschlag ist ein wenig umfangreicher als der zweite, der mit zwei Klassen weniger auskommt.

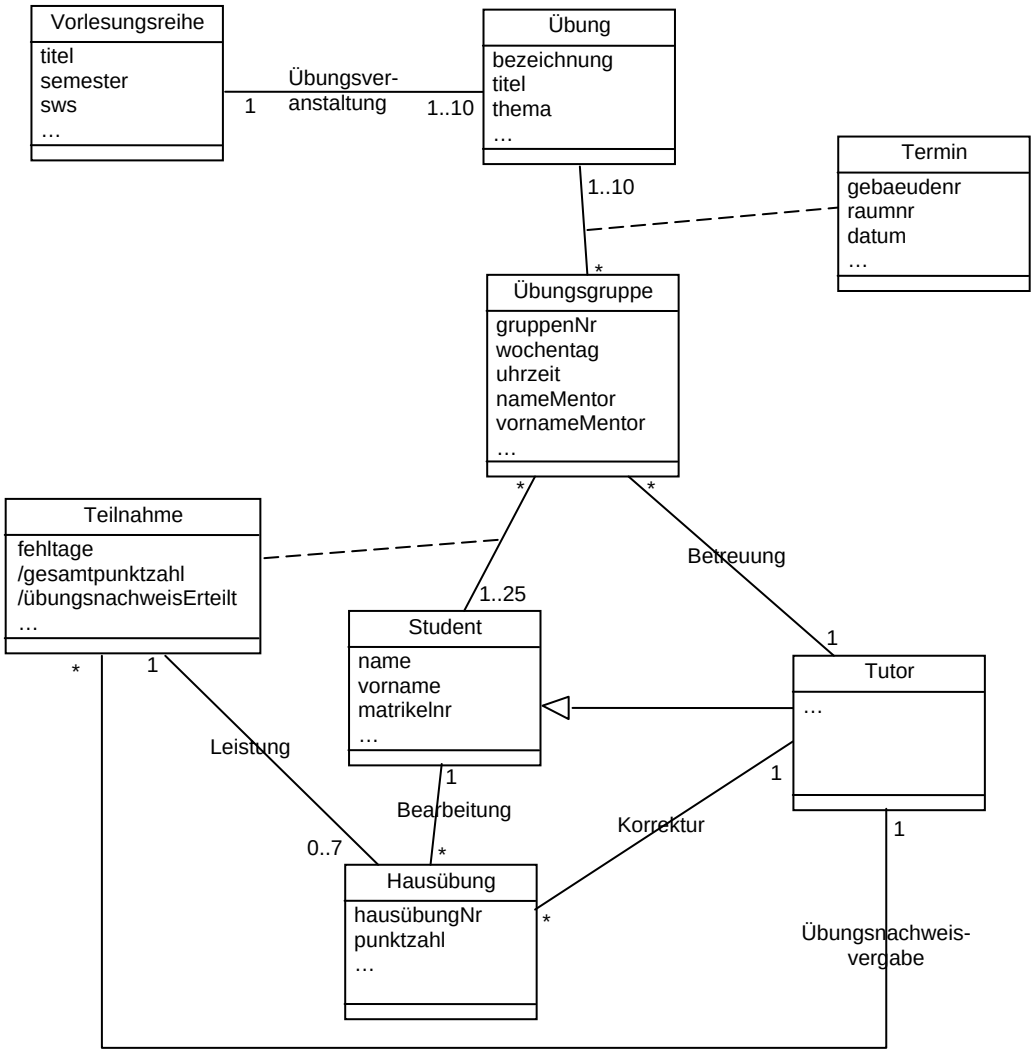



Abb. 1: Lösungsvorschlag 1 zur Aufgabe 1

Der Mentor tritt hier lediglich im Zusammenhang mit der Unterstützung des Tutors bei seiner Betreuungstätigkeit in Erscheinung. Zudem werden außer dem Namen keine weiteren Informationen über ihn benötigt, daher ist eine eigene Klasse für den Mentor nicht erforderlich (aber auch nicht falsch). Er wird über ein String-Attribut in der Klasse Übungsgruppe berücksichtigt. Klassen wie Lehrgebiet, Vorlesung, Semester oder Universität werden zur Lösung der Aufgabe nicht benötigt, daher werden für diese keine Punkte vergeben. Abzüge gibt es für Klassen, die eigentlich keine sind, wie z.B. Übungsgruppenliste.

An einigen Stellen hätte man auch stärkere Bindungen als Assoziationen verwenden können, z.B.:

- Komposition zwischen Vorlesungsreihe und Übung
- Aggregation zwischen Übungsgruppe und Student
- Aggregation zwischen Übungsgruppe und Tutor.

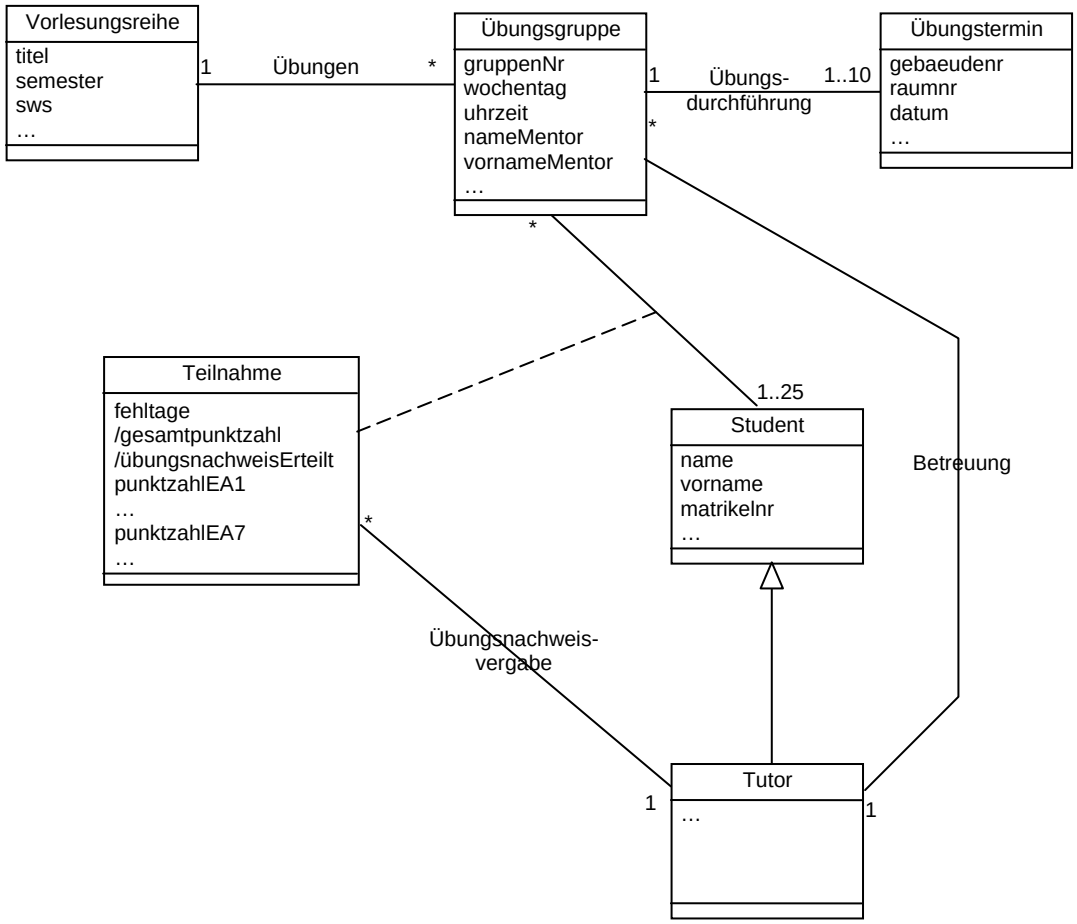



Abb. 2: Lösungsvorschlag 2 zur Aufgabe 1

Aufgabenteil c:
Folgende Einschränkungen in der Realität werden durch das Klassendiagramm in Abb. 1 nicht ausgedrückt:

- Ein Tutor muss in einem früheren Semester als Student Teilnehmer einer Übungsgruppe zu dieser oder einer vergleichbaren Vorlesungsreihe gewesen sein.
- Die Termine einer Übungsgruppe liegen im richtigen Semester, überschneiden sich nicht, etc.
- Ein Tutor betreut maximal eine Übungsgruppe pro Semester.
- Die Korrektur der Hausübungen und die Leistungsnachweisvergabe werden nur vom zuständigen Betreuer einer Übungsgruppe vorgenommen.
- …

## Aufgabe 2

a) Eine mögliche Lösung ist in Abb. 3 dargestellt.
Siehe auch Anhang, Abbildungen 8 und 10.

b) Eine mögliche Lösung ist in Abb. 6 dargestellt.
Siehe auch Anhang, Abbildungen 9 und 11.

c) Die in den Abbildungen 3 und 4 dargestellte Modellierung hat den Nachteil, dass die Schnittstellenklassen die „echten“ Entitätsobjekte kennen und folglich auch schreibend darauf zugreifen könnten. Dies ist jedoch nicht wünschenswert, weil eine Manipulation der Daten nur nach vorheriger Prüfung der Änderungen erfolgen sollte, was jedoch nicht in den Zuständigkeitsbereich der Schnittstellenklassen fällt. Dies ist Gegenstand der Anwendungslogik und somit der Kontrollklassen. Bei einer konsequenten Trennung der Verantwortlichkeiten im Sinn des Kurses schreibt ein Objekt einer Schnittstellenklasse Änderungen bzw. Benutzereingaben immer nur in Kopien von Entitätsobjekten. Die Verarbeitung dieser Eingaben und die Prüfung des Ergebnisses übernimmt dann das Objekt der zuständigen Kontrollklasse. Nur bei positivem Ergebnis der Prüfung werden dann die Entitätsobjekte und damit die Daten verändert (schreibender Zugriff auf die Entitätsobjekte).

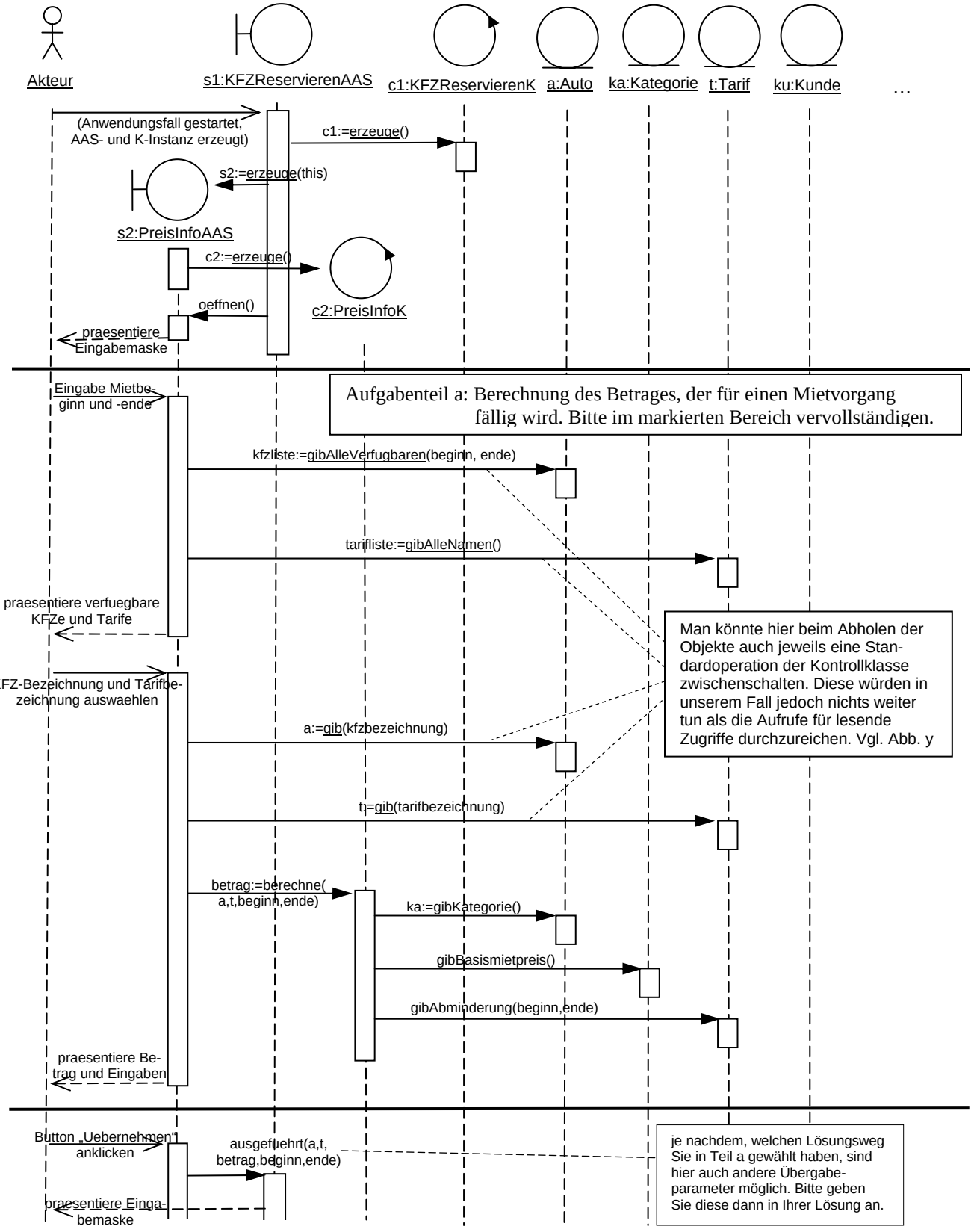



Abb. 3: Lösungsvorschlag zur Aufgabe 2a

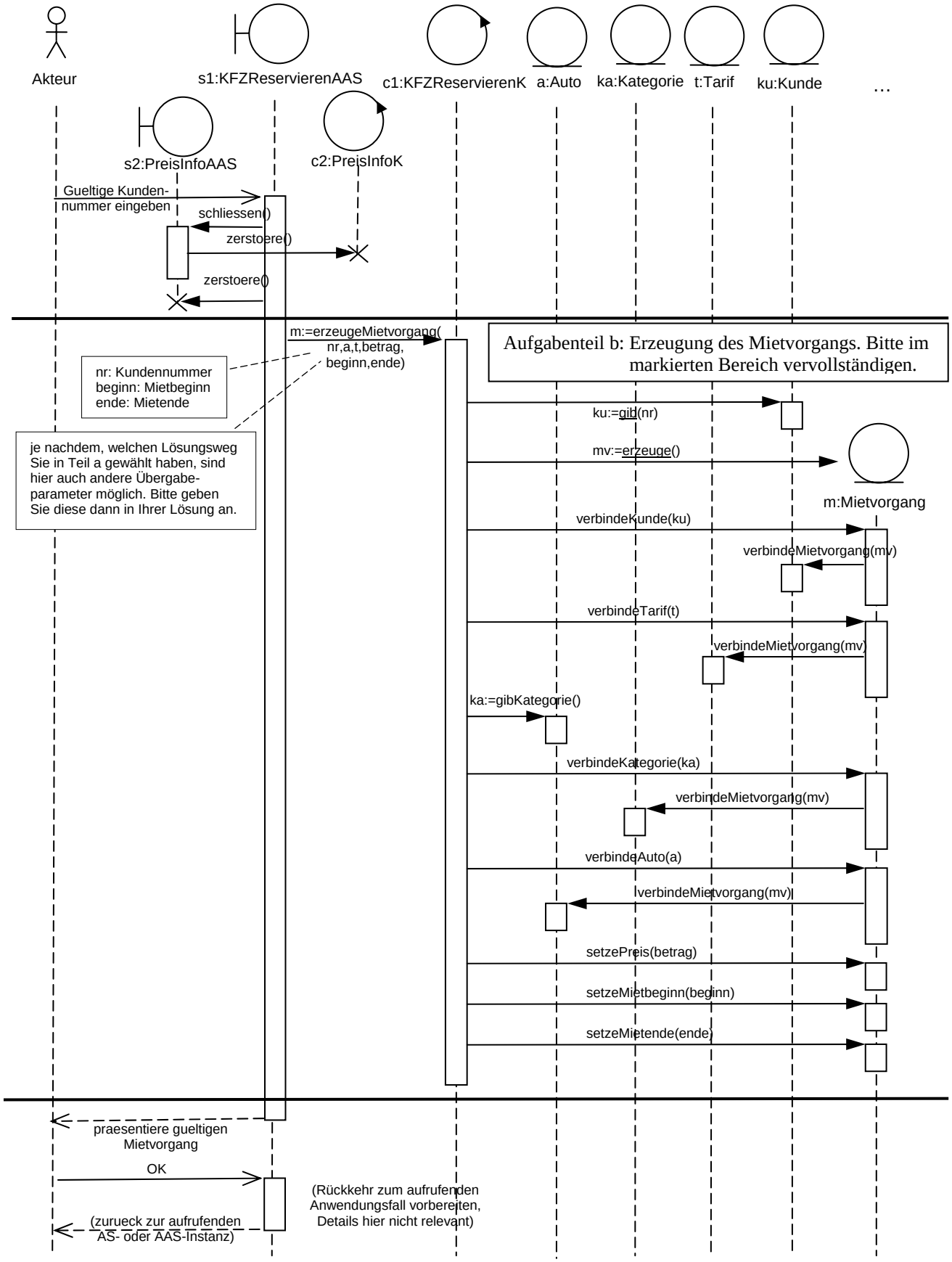



Abb. 4: Lösungsvorschlag zur Aufgabe 2b

## Aufgabe 3

Zwei mögliche Lösungen sind in den Abbildungen 5 und 6 dargestellt.

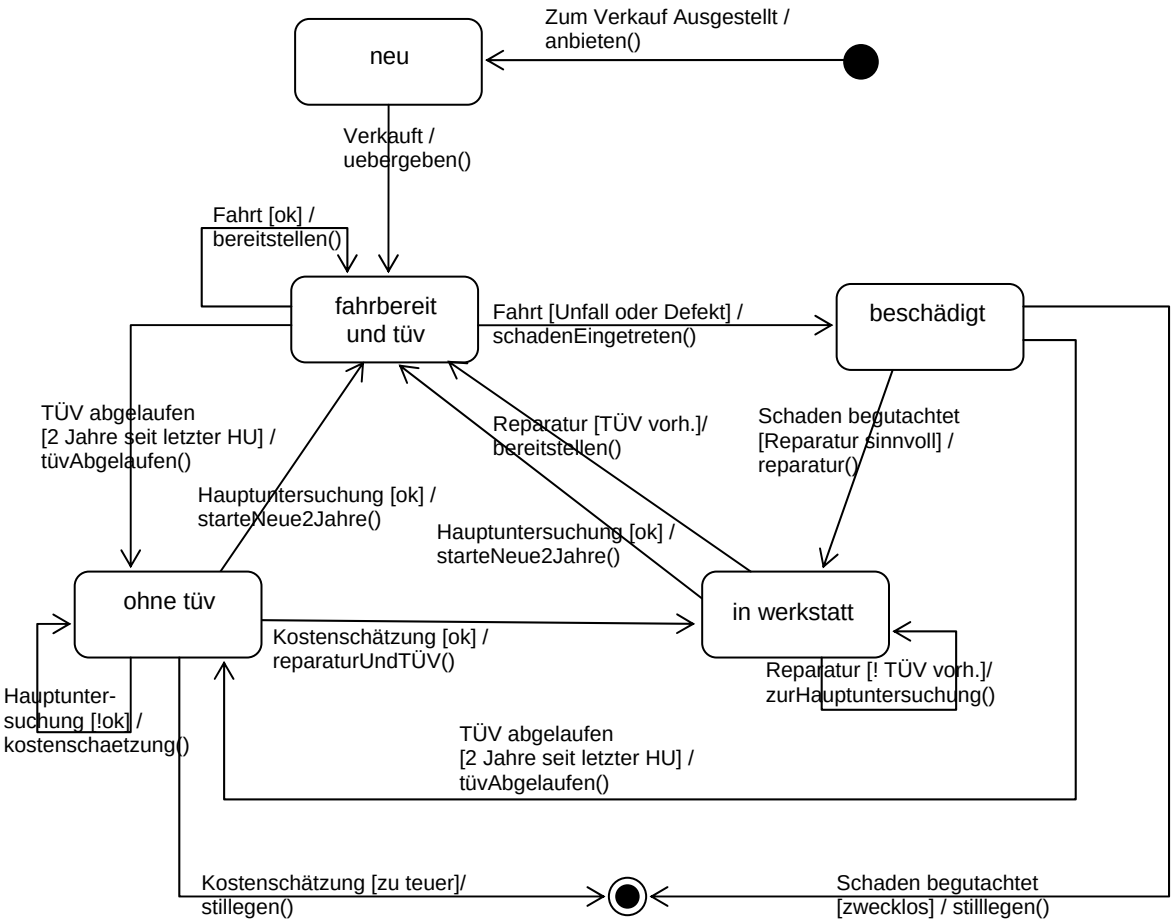



Abb. 5: Lösungsvorschlag 1 zur Aufgabe

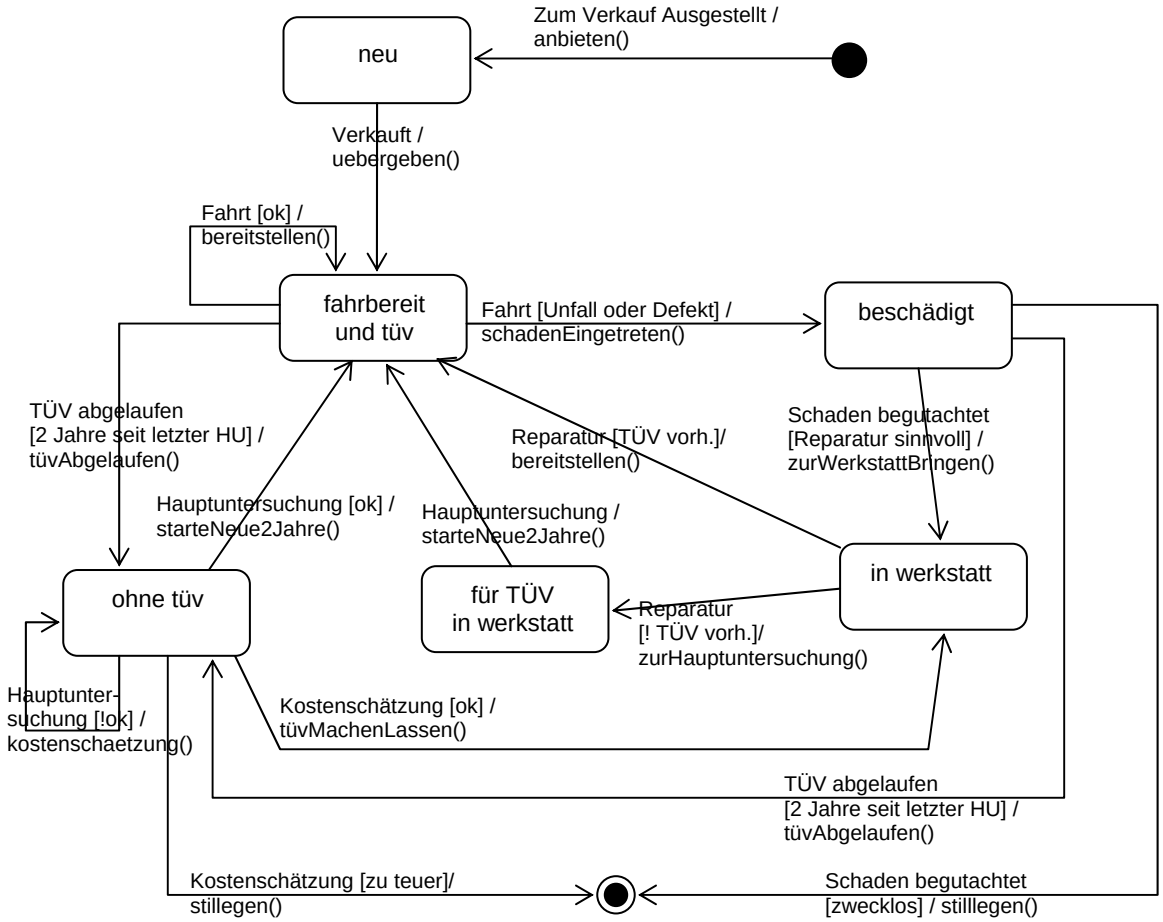



Abb. 6: Lösungsvorschlag 2 zur Aufgabe 3

## Aufgabe 4

Eine mögliche Lösung ist in Abb. 7 dargestellt. Für die Transformation der Assoziationen Team1, Team2 und Sieger könnte man auch drei separate Behälterklassen verwenden (SpielMengeTeam1, SpielMengeTeam2, SpielMengeSieger). Da sich diese Klassen jedoch durch nichts unterscheiden, reicht eine Klasse SpielMenge aus. Die Multiplizität bei der Klasse SpielMenge ist 2..3, weil jedes Spiel ein Team1 und ein Team2 hat (und folglich in den entsprechenden Mengen der jeweiligen Teams enthalten ist), aber nicht unbedingt einen Sieger.

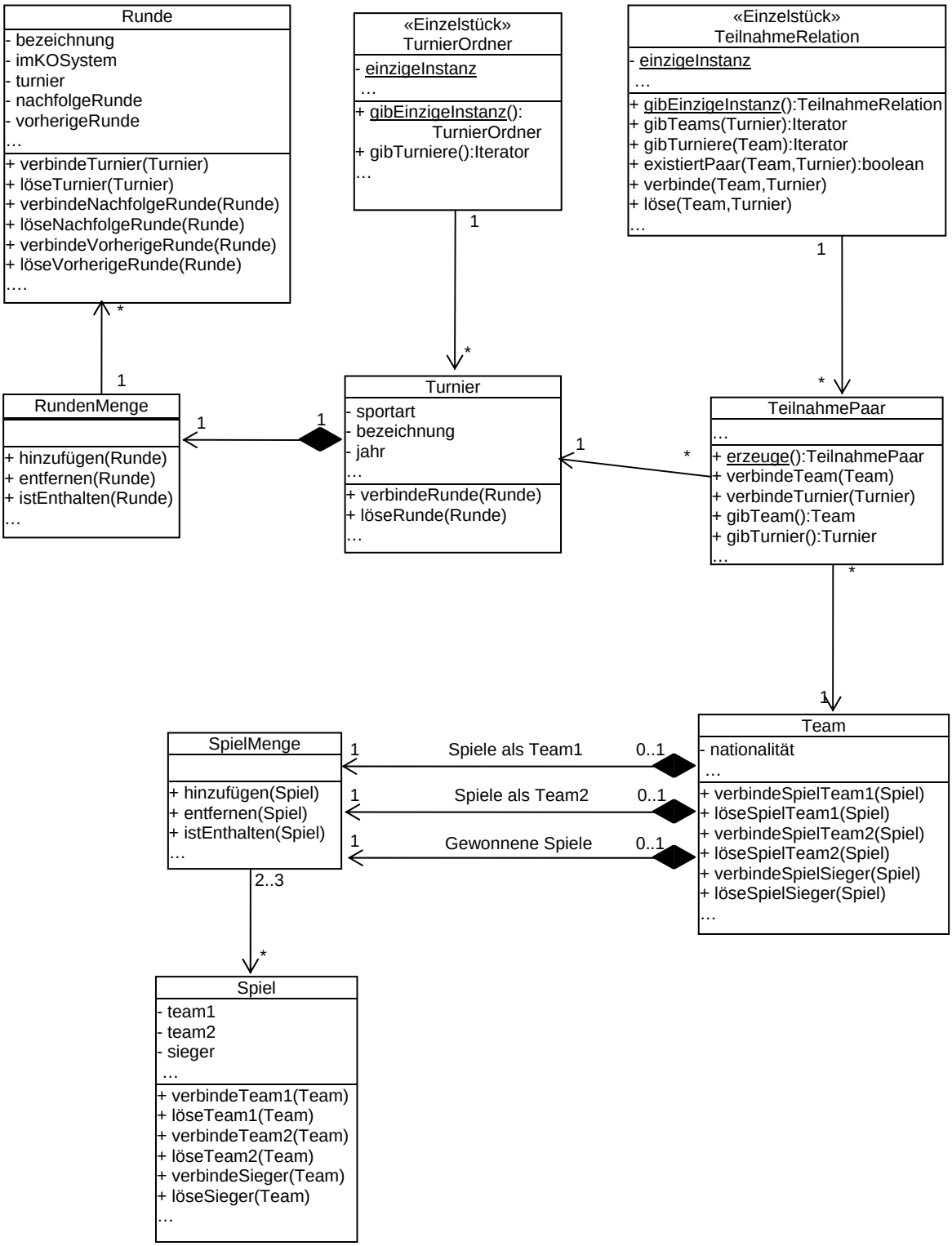



Abb. 7: Lösungsvorschlag 1 zur Aufgabe 4

## Anhang

Die folgenden Abbildungen zeigen Lösungsmöglichkeiten für die Aufgabe 2 unter der Voraussetzung, dass auf Kopien gearbeitet werden soll. Dies war in der Aufgabenstellung nicht verlangt, die Abbildungen dienen daher nur zur Information.

Die erste Variante (Abb. 8 und 9) ist stark an den Abbildungen 3 und 4 angelehnt, also dem Lösungsweg beim Arbeiten ohne Kopien. Interessanter und näher an der Realität ist der zweite Lösungsvorschlag (Abb. 10 und 11). Hierbei wird schon für den Berechnungsvorgang ein transienter Mietvorgang erzeugt, also eine Kopie. Dadurch spart man sich die umfangreichen und unübersichtlichen Übergabeparameter, was unter anderem die Fehleranfälligkeit und den Änderungsaufwand verringert und den Vorgang transparenter macht. Bei der Erzeugung des „richtigen“, also des persistenten Mietvorgangs, müssen dann im Wesentlichen nur noch die Assoziationen und die Attribute von der Kopie auf das Entitätsobjekt übertragen werden.

Infolge des Arbeitens auf Kopien werden Sequenz- und Kollaborationsdiagramme sehr umfangreich und unübersichtlich. In der Praxis wird man daher im Grobentwurf solche Modellierungen nur für wenige ausgewählte Anwendungsfälle vornehmen und bei den restlichen von den Details der Verwendung von Kopien abstrahieren.

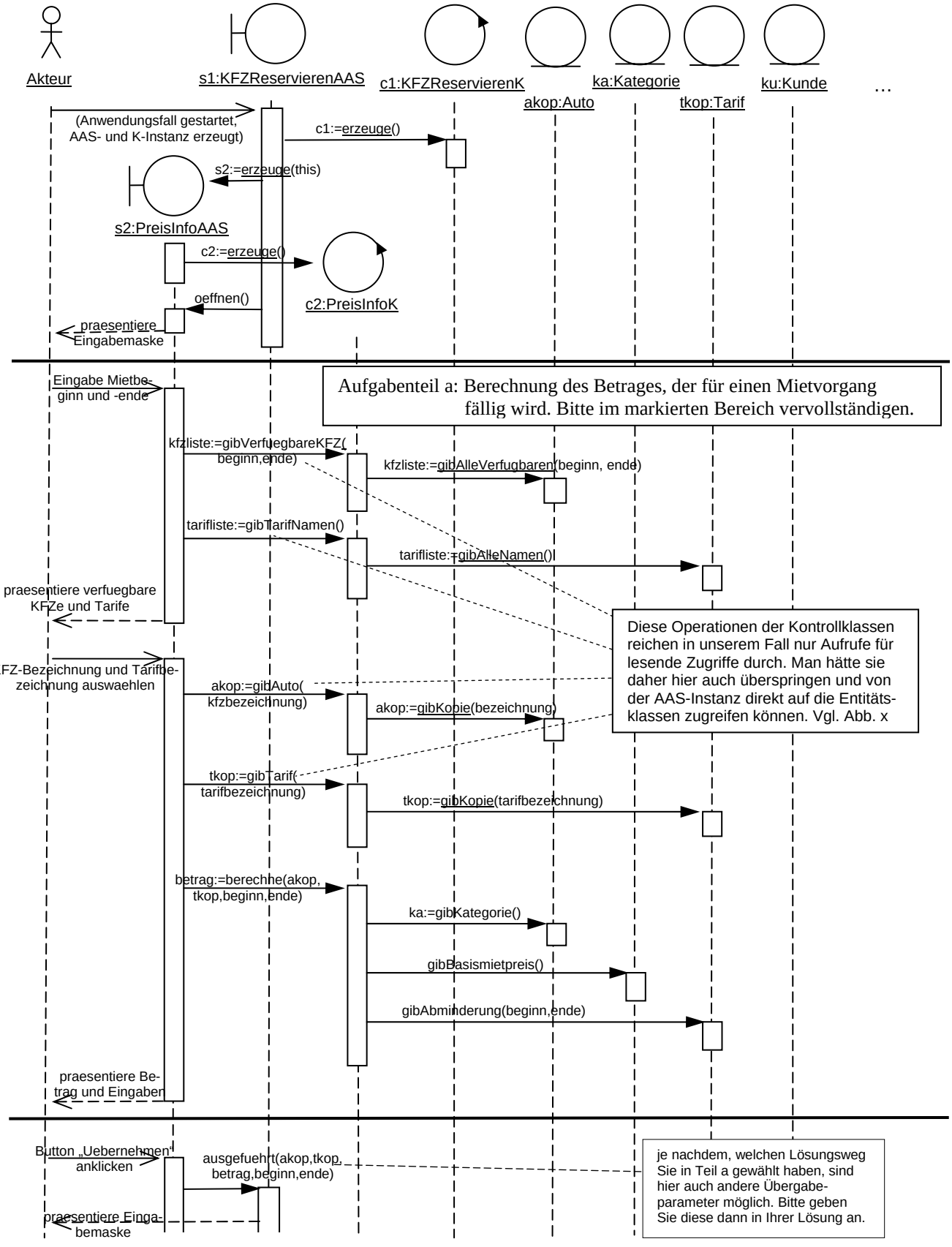



Abb. 8: Lösungsvorschlag 1 zu Aufgabe 2a bei Verwendung von Kopien
(war nicht verlangt, nur zur Information)

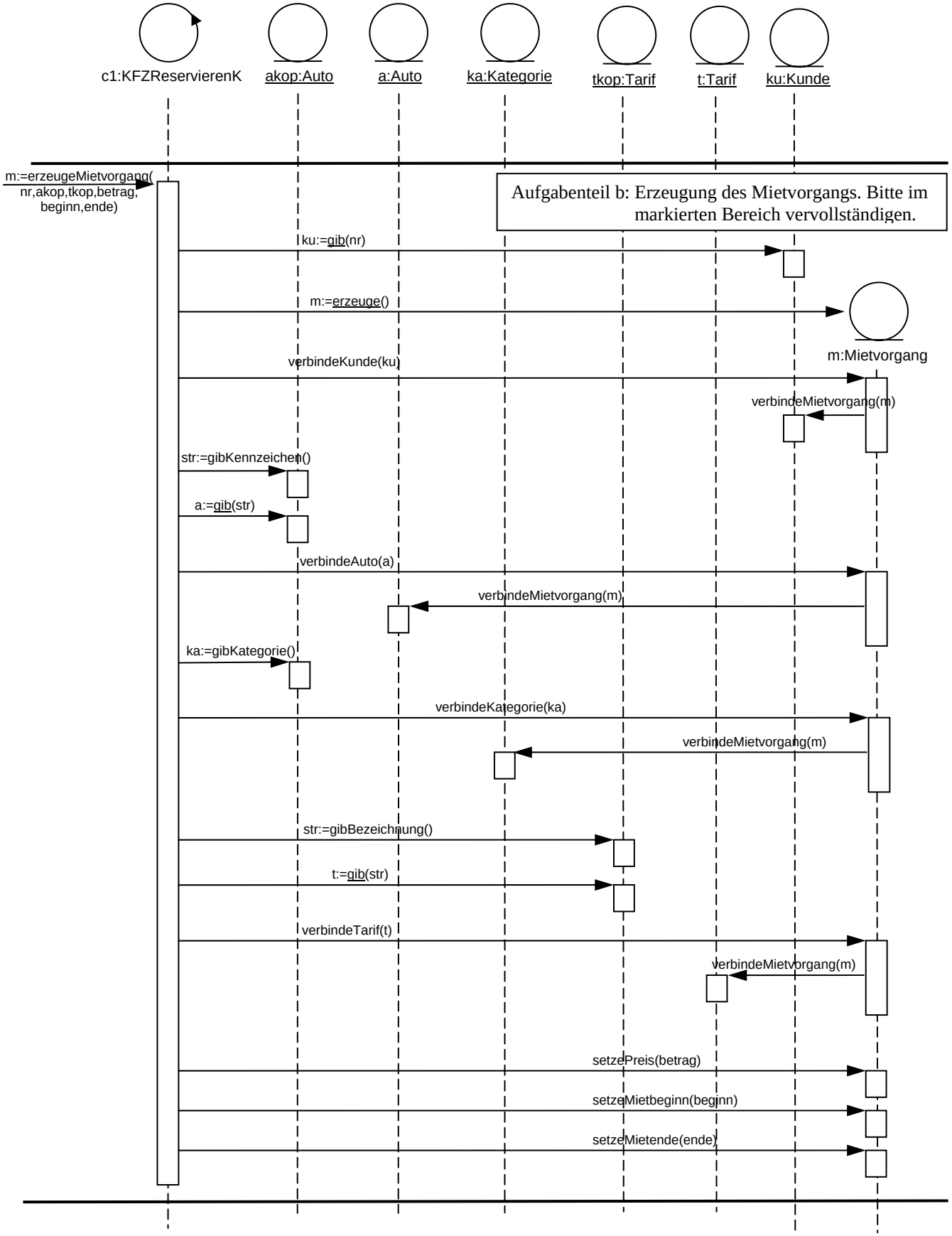
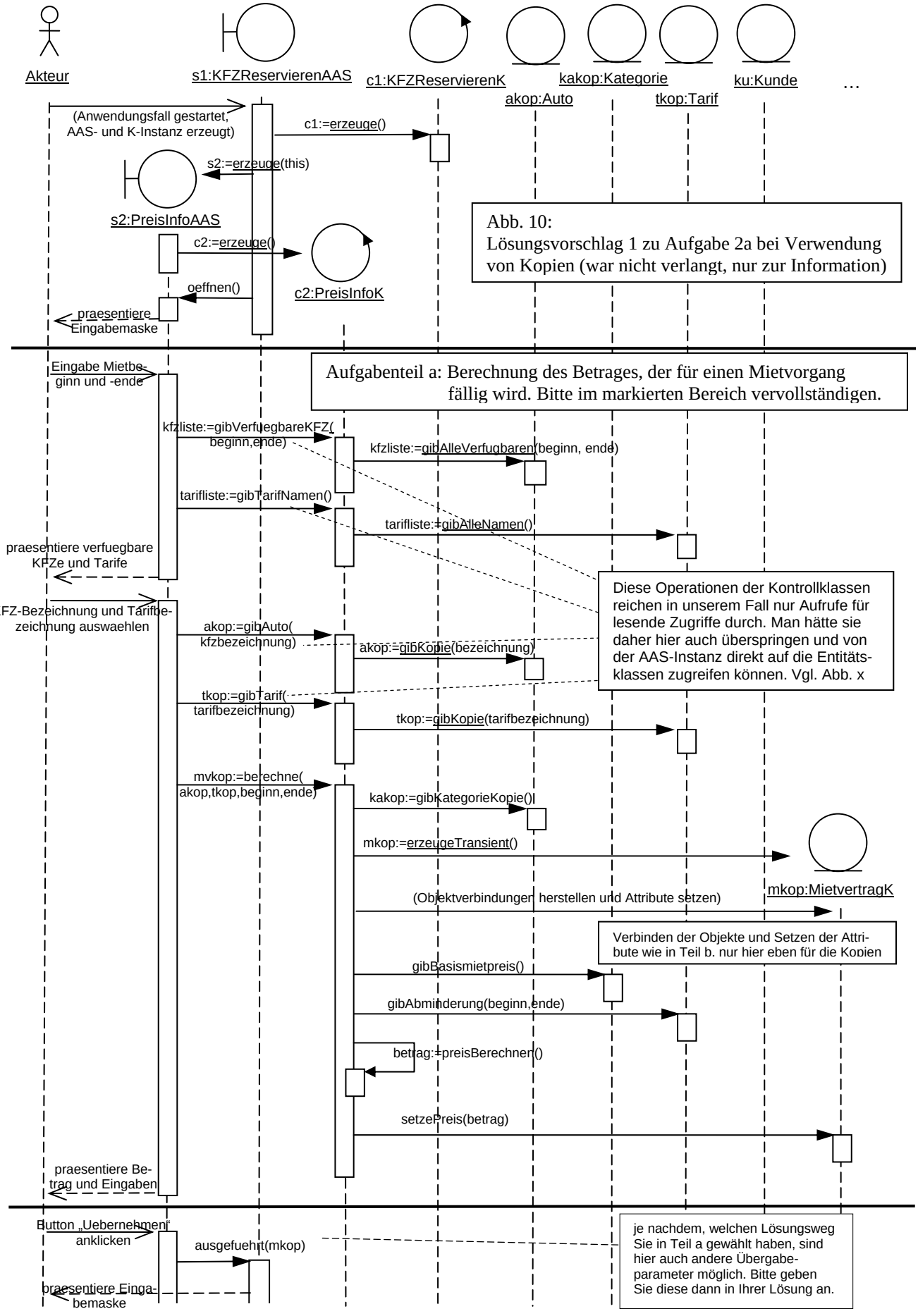



Abb. 9: Lösungsvorschlag 2 zu Aufgabe 2b bei Verwendung von Kopien (war nicht verlangt, nur zur Information)


Akteur
s1:KFZReservierenAAS
c1:KFZReservierenK
akop:Auto
kakop:Kategorie
tkop:Tarif
ku:Kunde
...
(Anwendungsfall gestartet, AAS- und K-Instanz erzeugt)
c1:=erzeuge()
s2:=erzeuge(this)
s2:PreisInfoAAS
c2:=erzeuge()
c2:PreisInfoK
oeffnen()
praesentiere Eingabemaske
Abb. 10:
Lösungsvorschlag 1 zu Aufgabe 2a bei Verwendung von Kopien (war nicht verlangt, nur zur Information)
Eingabe Mietbeginn und -ende
Aufgabenteil a: Berechnung des Betrages, der für einen Mietvorgang fällig wird. Bitte im markierten Bereich vervollständigen.
kfzliste:=gibVerfuegbareKFZ(beginn,ende)
kfzliste:=gibAlleVerfugbaren(beginn, ende)
tarifliste:=gibTarifNamen()
tarifliste:=gibAlleNamen()
praesentiere verfuegbare KFZe und Tarife
Diese Operationen der Kontrollklassen reichen in unserem Fall nur Aufrufe für lesende Zugriffe durch. Man hätte sie daher hier auch überspringen und von der AAS-Instanz direkt auf die Entitätsklassen zugreifen können. Vgl. Abb. x
KFZ-Bezeichnung und Tarifbezeichnung auswaehlen
akop:=gibAuto(kfzbezeichnung)
akop:=gibKopie(bezeichnung)
tkop:=gibTarif(tarifbezeichnung)
tkop:=gibKopie(tarifbezeichnung)
mvkop:=berechne(akop,tkop,beginn,ende)
kakop:=gibKategorieKopie()
mkop:=erzeugeTransient()
mkop:MietvertragK
(Objektverbindungen herstellen und Attribute setzen)
Verbinden der Objekte und Setzen der Attribute wie in Teil b, nur hier eben für die Kopien
gibBasismietpreis()
gibAbminderung(beginn,ende)
betrag:=preisBerechnen()
setzePreis(betrag)
praesentiere Betrag und Eingaben
Button „Uebernehmen" anklicken
ausgefuehrt(mkop)
praesentiere Eingabemaske
je nachdem, welchen Lösungsweg Sie in Teil a gewählt haben, sind hier auch andere Übergabeparameter möglich. Bitte geben Sie diese dann in Ihrer Lösung an.

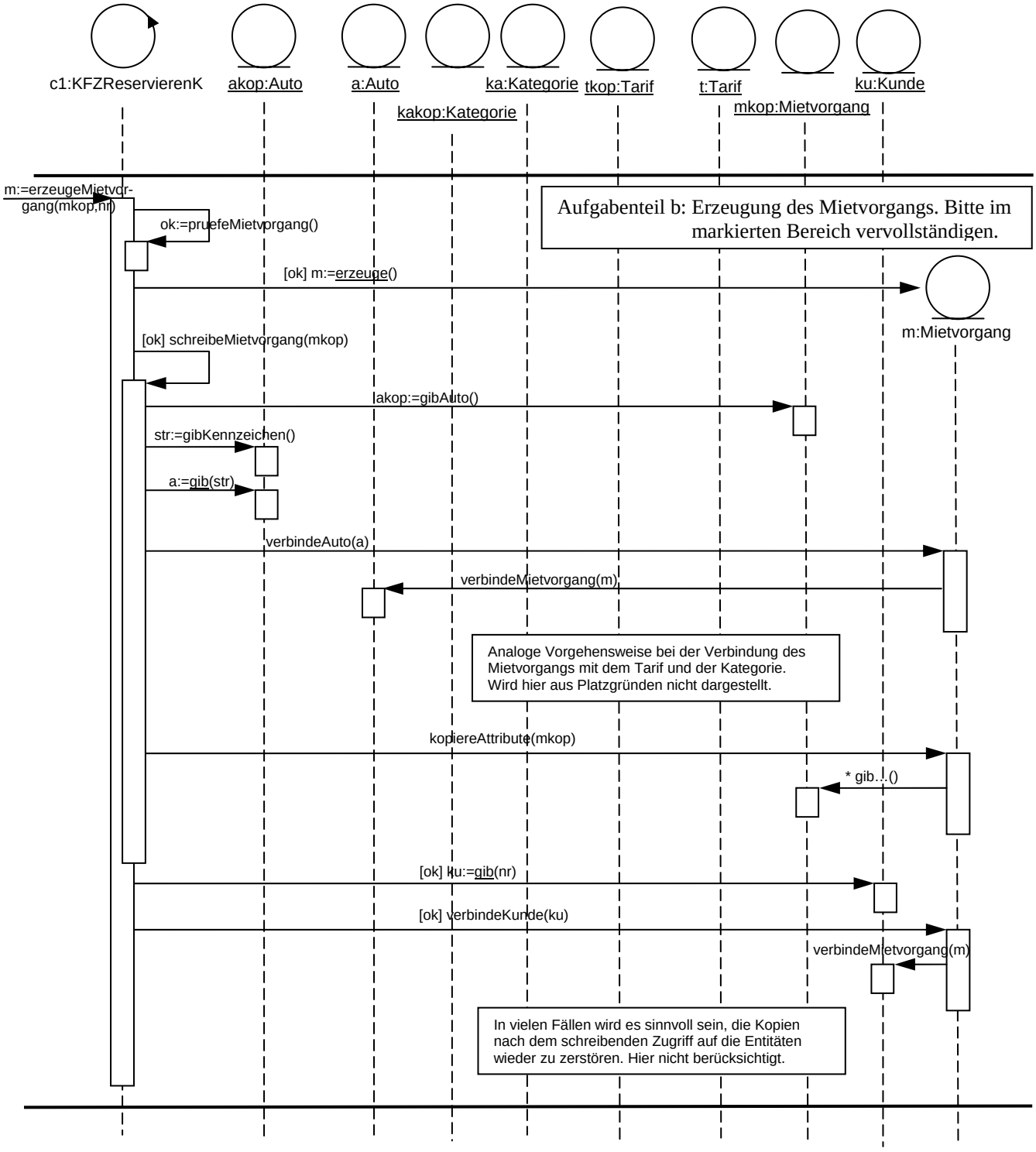



Abb. 11: Lösungsvorschlag 3 zu Aufgabe 2b bei Verwendung von Kopien
(war nicht verlangt, nur zur Information)