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

Herrmann
Jucknath
Vösgen

Klausur

Softwaretechnik

23. Februar 2005

Prüfen Sie zunächst, ob dieses Exemplar vollständig ist (15 Blätter mit Aufgaben ohne Deckblatt und 3 zusätzliche Blätter). 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/Prüfungsordnung: ______________
SWT-Übungen habe ich im Semester ________ besucht.

Aufgabe	1	2	3	Gesamt
Maximal:	40	40	20	100
Erreicht:				

Aufgabe 1 (Fusion) - Zeitarbeitsfirma

(Geschätzte Bearbeitungszeit: 50 min.) Eine Zeitarbeitsfirma gibt ein Verwaltungssystem zur Unterstützung ihrer Geschäftsprozesse in Auftrag.

Die Geschäftsidee der Zeitarbeitsfirmen läßt sich auf eine einfache Formel bringen: In einem Unternehmen wird für eine befristete Zeit Personal gebraucht. Sei es, weil ein Mitarbeiter krank ist oder weil das Unternehmen Verstärkung benötigt, um einen Liefertermin einhalten zu können. Die Zeitarbeitsfirma stellt Zeitarbeiter leihweise zur Verfügung und erhält dafür ein entsprechendes Entgelt.

Ein Unternehmer gibt der Zeitarbeitsfirma zunächst einen Auftrag, in dem die Anforderungen beschrieben sind. Erst wenn klar ist, ob die Zeitarbeitsfirma diesen Auftrag erfüllen kann, wird daraus ein Vertrag erstellt. Wenn hierbei für den Unternehmer Unklarheiten auftreten, kann einer von 5 Sachbearbeitern aus der Verwaltung (der Zeitarbeitsfirma) weiterhelfen.

In einem Auftrag werden die Bedingungen festgelegt, mindestens Beginn und Ende des Einsatzes, benötigte Zahl der Zeitarbeiter, geforderte Qualifikation und ein Kostenvorschlag. Die verschiedenen Qualifikationen werden in einem Profil zusammengefasst. Über eine Website können allgemein alle verfügbaren Profile anzeigt werden, aber auch Profile einzelner Zeitarbeiter. Jeder Zeitarbeiter kann ein oder mehrere Profile besitzen. (Beispiel für ein Profil, welches zwei Qualifikationen zusammenfasst: *Java-Programmierer mit Datenbankkentnissen*).

Ein Zeitarbeiter ist entweder bei der Zeitarbeitsfirma angestellt oder wird als Freiberufler von der Zeitarbeitsfirma vermittelt. Wenn er ein Angestellter ist, kann er über eines von 42 Terminals seine eigenen Daten aktualisieren oder sich über aktuelle Aufträge selbst informieren. Ist er ein Freiberufler, so wird er zwar vermittelt und seine Daten werden von einem Sachbearbeiter auf die Website gestellt, er selbst hat aber keinen direkten Zugriff auf das System.

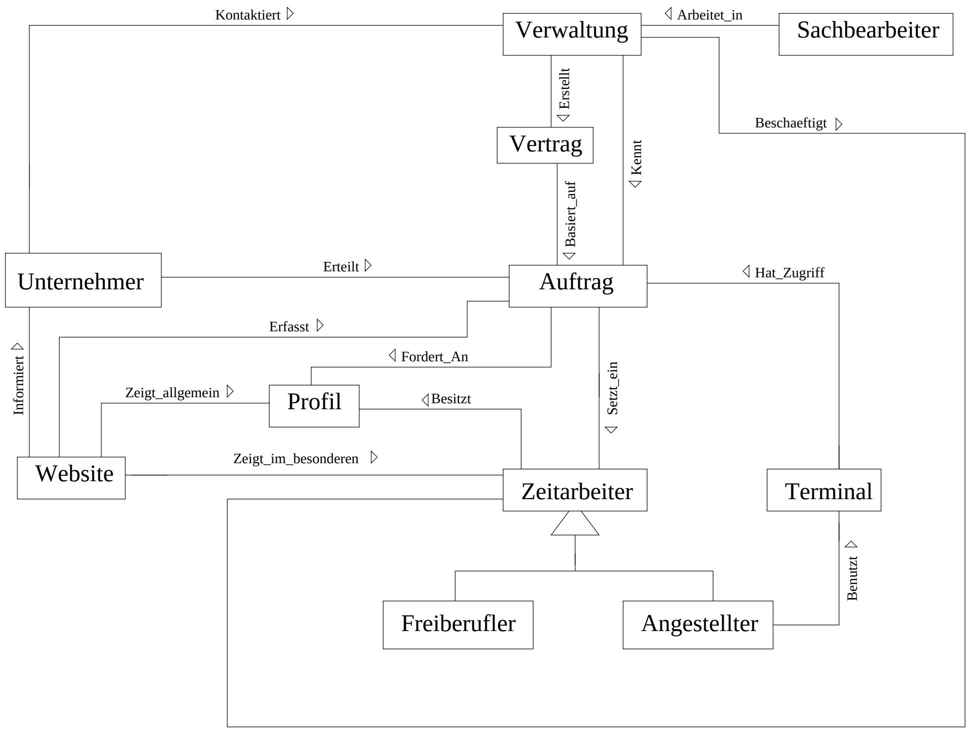
Für den Unternehmer bietet die Website die Möglichkeit sich zu informieren, welche Art von Profilen von der Zeitarbeitsfirma allgemein angeboten werden und wieviele Zeitarbeiter mit den entsprechenden Profilen im besonderen vermittelt werden können. Darüber hinaus kann der Unternehmer auf der Website einen Auftrag eingeben und direkt kontrollieren, ob dieser auch wunschgemäß angezeigt wird.

Aufgabenstellung

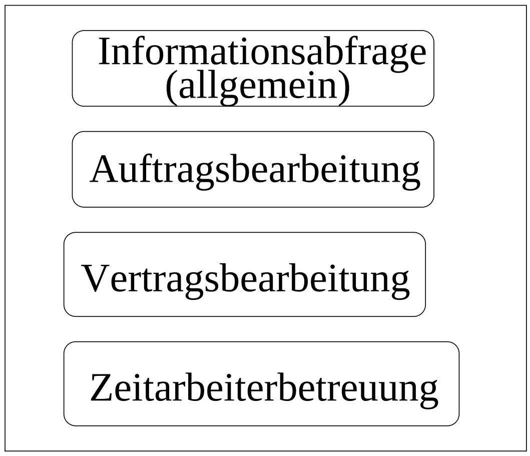
Ein System zur Verwaltung der Zeitsarbeitsfirma soll mit der Fusion-Methode entwickelt werden. Dabei soll das Verhalten der Systemoperationen $auftrag\_eingeben$ und $auftrag\_suchen$ besonders genau untersucht werden.

1. Geben Sie die *Multiplizitäten* im gegebenen Klassenmodell an. *4 Punkte*
2. Vervollständigen Sie das gegebene Use-Case-Modell, indem Sie die Akteure des Systems festlegen und sie den Use Cases zuordnen. *3 Punkte*
3. Vervollständigen Sie das vorgegebene Systemklassenmodell, indem Sie die Systemgrenze festlegen und gegebenenfalls weitere neue Klassen und Assoziationen hinzufügen.
   Hinweis: Die Multiplizitäten sollen in das Systemklassenmodell nicht übertragen werden. *6 Punkte*
4. Vervollständigen Sie das *Operationsschema* für die Systemoperation $auftrag\_suchen$. *15 Punkte*
5. Stellen Sie den Objektinteraktionsgraphen auf für die Systemoperation $auftrag\_eingeben$. *12 Punkte*

Klassenmodell

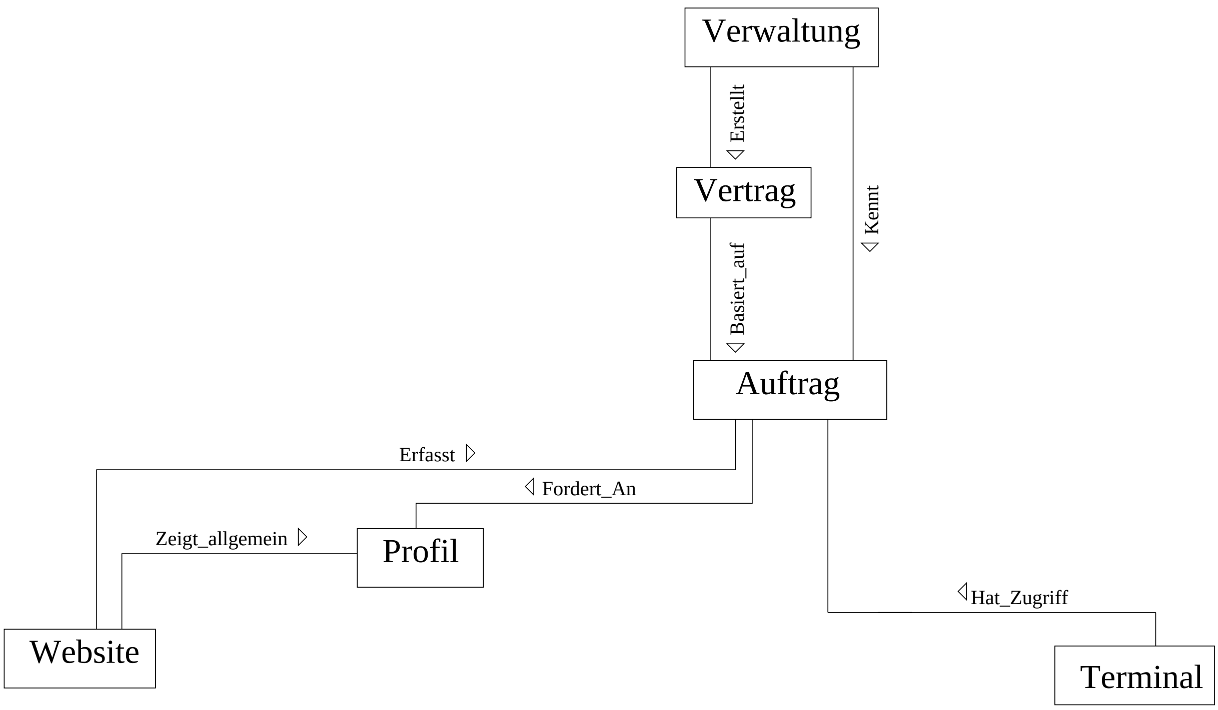


Use-Case-Modell



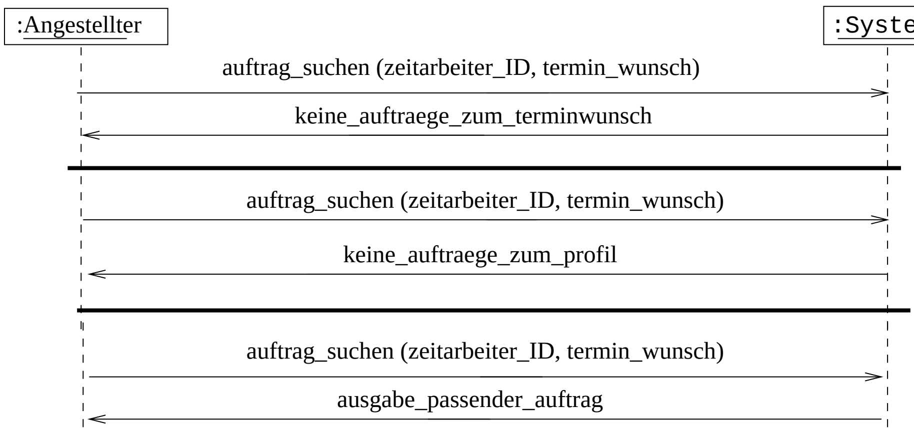
Auftragsbearbeitung = Auftragseingabe und Auftragsannahme

Systemklassenmodell



Operationsschema

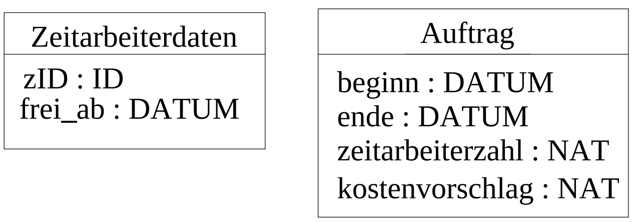
Die Systemoperation $auftrag\_suchen$ wird von einem Angestellten der Zeitsarbeitsfirma angestossen, der für sich selbst einen Auftrag sucht und sich im Erfolgsfall direkt einträgt. Berücksichtigen Sie bei der Spezifikation die im folgenden Time-Line-Diagramm aufgeführten **Fehlerfälle**.



Sie können für den Vergleich zweier Termine die Funktion $diff\_datum$ verwenden, die zwei Parameter als Eingabe benötigt und als Ausgabe -1 liefert, wenn der erste Termin vor dem zweiten Termin liegt, 0 wenn beide Termine identisch sind und 1 sonst.

Folgende **Klassenattribute** sollen in der Spezifikation des Operationsschemas verwendet werden. Das Attribut $frei\_ab$ gibt an, ab wann der Zeitarbeiter wieder vermittelt werden kann.

Beachten Sie bitte: das Operationsschema muß zu Ihrem Systemklassenmodell konsistent sein!



Op. = auftrag_suchen

Desc. = Ein Angestellter der Zeitarbeitsfirma sucht für sich selbst einen neuen Auftrag, der zu seinem Terminwunsch und zu seinem Profil passt. Wird ein passender Auftrag gefunden, soll der Zeitarbeiter direkt für diesen Auftrag festgelegt werden.

Input = $zeitarbeiter\_ID : ID,\ termin\_wunsch : DATUM$

Reads = $za : Zeitarbeiter$ **with** $za.id = zeitarbeiter\_id$
$t : Terminal,\ Profil,\ Auftrag,\ Vertrag$
$Benutzt,\ Besitzt,\ Fordert\_an$

Changes = $Setzt\_ein$

Sends =

Pre =

Post = **let**

$auftraege\_zum\_termin ==$

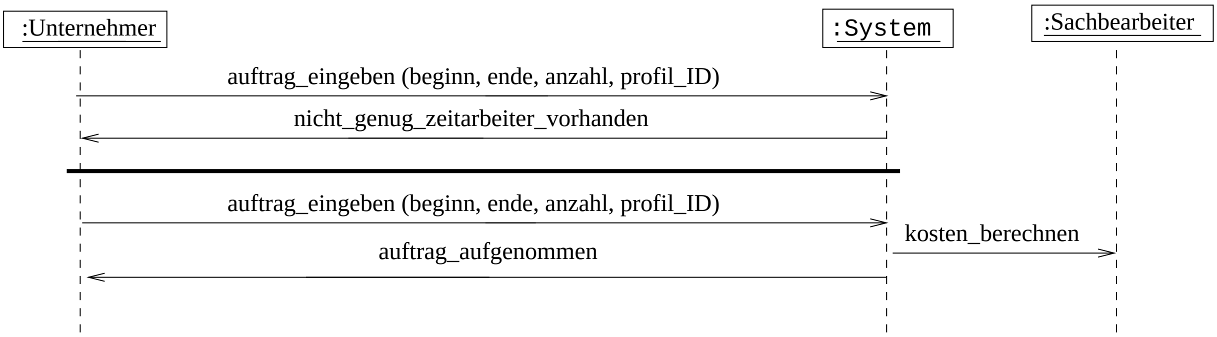
$passende\_profile ==$

$\bullet$

$(auftraege\_zum\_termin = \varnothing \Rightarrow$
$\quad is\_sent\{keine\_auftraege\_zum\_terminwunsch\}) \wedge$
$(passende\_profile = \varnothing \Rightarrow$
$\quad is\_sent\{keine\_auftraege\_zum\_profil\}) \wedge$
$(auftraege\_zum\_termin \neq \varnothing \wedge passende\_profile \neq \varnothing \Rightarrow$

Vorgaben zum Objektinteraktionsgraph

Zur Spezifikation der Systemoperation $auftrag\_eingeben$ benötigen Sie:



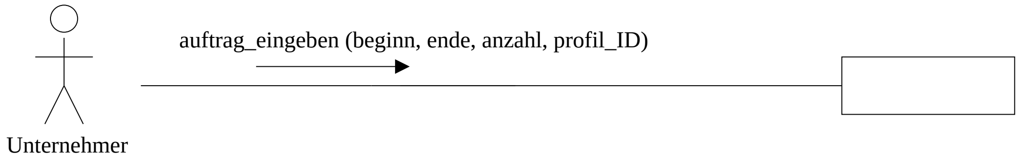
Ein Unternehmer gibt einen Auftrag im System ein. Wenn nicht genügend Zeitarbeiter mit diesem Profil frei sind, dann soll dem Unternehmer eine entsprechende Meldung gemacht werden. Sonst soll dem Unternehmer gemeldet werden, daß der Auftrag aufgenommen ist und einem Sachbearbeiter soll gemeldet werden, daß zu diesem Auftrag ein Kostenvorschlag berechnet werden muß. Es wird in jedem Fall ein neuer Auftrag angelegt.

Op.	=	auftrag_eingeben
Desc.	=	Ein Unternehmer gibt einen Auftrag im System ein.
Input	=	$beginn : DATUM, ende : DATUM, anzahl : NAT, profil\_ID : ID$
Reads	=	w : *Website*, *Zeitarbeiterdaten*, v : *Verwaltung*, p : *Profil* with $p.ID = profil\_ID$, *Besitzt*, *Erfasst*
Changes	=	a : *Auftrag* type, *Auftrag*, *Kennt*, *Fordert_an*
Sends	=	$Unternehmer : \{nicht\_genug\_zeitarbeiter\_vorhanden,$ $auftrag\_aufgenommen\},$ $Sachbearbeiter : \{kosten\_berechnen\}$
Pre	=	$-$
Post	=	**let** $passende\_zeitarbeiter == \{z : Zeitarbeiter \mid$ $(z, p) \in Besitzt \wedge diff\_datum(z.frei\_ab, beginn) \leq 0\}$ $passt == \#(passende\_zeitarbeiter) \geq anzahl$ $\bullet$ $(a : Auftrag$ new $\wedge\ a.beginn = beginn \wedge a.ende = ende \wedge$ $a.zeitarbeiterzahl = anzahl \wedge Auftrag' = Auftrag \cup \{a\}$ $\wedge\ Kennt' = \{(v, a)\} \cup Kennt \wedge Fordert\_an' = \{(a, p)\} \cup Fordert\_an$ $\wedge\ (\neg passt \Rightarrow is\_sent\{nicht\_genug\_zeitarbeiter\_vorhanden\})$ $\wedge\ (passt \Rightarrow is\_sent\{auftrag\_aufgenommen\}$ $\wedge\ is\_sent\{kosten\_berechnen\}))$

Objektinteraktionsgraph

Entwickeln Sie auf Basis des vorgegebenen Timeline-Diagramms und Operationsschema den Objektinteraktionsgraph für die Systemoperation $auftrag\_eingeben$.

Beachten Sie bitte: der Objektinteraktionsgraph muß zu Ihrem Systemklassenmodell konsistent sein!



Aufgabe 2 (Object-Z) - Read-Only-Dateisystem

Ein Dateisystem, in dem nur gelesen werden darf, soll in Object-Z modelliert werden.

Wie in den meisten Dateisystemen üblich, werden die Daten in Dateien abgespeichert und die Dateien in baumartig verschachtelten Verzeichnissen angeordnet. Zugriffsrechte beschränken den Zugriff. Die Information, wer auf eine Datei zugreifen darf, wird in einer Rechte-Datenbank gespeichert.

Struktur der Modellierung

Aufbau des Baumes

Die Wurzel des Verzeichnisbaumes in *Dateisystem* bildet ein *Verzeichnis* namens *wurzel*; der Rest des Aufbaus ist beliebig, solange die Baum-Eigenschaften nicht verletzt werden. Das bedeutet: Es gibt nur eine Wurzel und keine Zyklen.

Das Dateisystem wird durch die Menge seiner gültigen *Pfad*e beschrieben. Ein gültiger *Pfad* im Dateisystem führt von der *wurzel* aus den Baum herab bis zu einem beliebigen Knoten. Alle *Pfad*e sind vom Typ injektive, nicht-leere Sequenz, in Object-Z $iseq_1$. Die Menge *pfade* enthält die gültigen *Pfad*e.

Ausserdem besitzt jedes *Verzeichnis* eine Menge *unterverzeichnisse*, in der alle direkten Unter-*Verzeichnis*se enthalten sind.

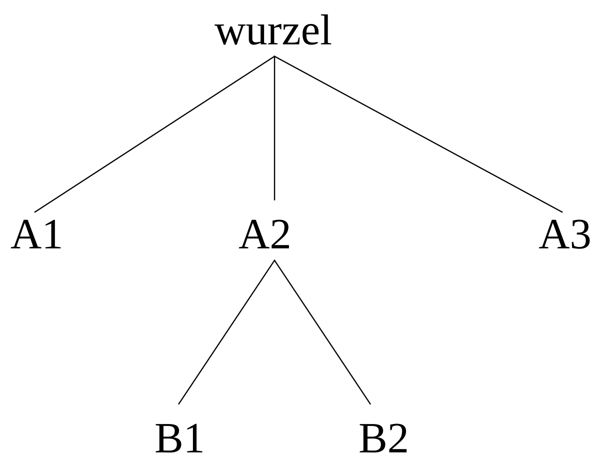


Abbildung 1: Gültiger Baum

Im obigen Beispiel gilt z.B.:

$$pfade = \{< wurzel >, < wurzel, A1 >, < wurzel, A2 >, < wurzel, A2, B1 >,$$
$$< wurzel, A2, B2 >, < wurzel, A3 >\}$$

und

$$wurzel.unterverzeichnisse = \{A1, A2, A3\} \wedge A2.unterverzeichnisse = \{B1, B2\} \wedge (\ldots)$$

Zugriffsrechte

Die Zugriffsrechte der Benutzer befinden sich im Objekt *rechteDB* der Klasse *Rechteverwaltung*.

Aufgabe: Spezifizieren Sie das *Dateisystem* in Object-Z, indem Sie die vorgegebenen Klassendefinitionen ergänzen. Sie müssen die vorgegebenen Signaturen der Operationen benutzen und dürfen keine neuen Operationen definieren.

Die Aufgabe gliedert sich in die Modellierung der Klassen

- Verzeichnis — *6 Punkte*
- Rechteverwaltung — *11 Punkte*
- Dateisystem — *17 Punkte*

und einer Untersuchung der Invariante von *Dateisystem*. — *6 Punkte*

(Geschätzte Bearbeitungszeit: 50 min.)

Vorgaben

Übernehmen Sie diese Vorgaben unverändert:

[Datensatz, Nutzer]
Nachricht ::= ok|datei_nicht_in_verzeichnis|pfad_nicht_im_system|
zugriff_nicht_gestattet
Pfad == $iseq_1$ Verzeichnis

Datei

Datei

$\upharpoonright (dat)$

$dat : Datensatz$

Teilaufgaben

Verzeichnis

Verzeichnisse enthalten Dateien und Unterverzeichnisse. Modellieren Sie diese Operationen:

1. *daten_liefern* ist nur anwendbar, wenn sich die übergebene *Datei* im *Verzeichnis* befindet. Sie liefert den *Datensatz* der *Datei* und kommuniziert mit einer Ausgabevariable nach außen, dass sich die Datei im Verzeichnis befindet. *2 Punkte*

2. *datei_nicht_im_verzeichnis* ist anwendbar, wenn *daten_liefern* nicht anwendbar ist. Sie muss das Nicht-Vorhandensein der *Datei* nach außen kommunizieren. *2 Punkte*

3. *in_verzeichnis_lesen* kombiniert die beiden anderen Operationen der Klasse durch Schemakalkül so, dass das System eine Auswahl zwischen *daten_liefern* und *datei_nicht_im_verzeichnis* treffen muss. *2 Punkte*

Verzeichnis

$\upharpoonright (in\_verzeichnis\_lesen)$

$dateien : \mathbb{F}\, Datei$

$unterverzeichnisse : \mathbb{F}\, Verzeichnis$

datei_nicht_im_verzeichnis

$f? : Datei$

$datei\_im\_verzeichnis! : \mathbb{B}$

$dat! : Datensatz$

daten_liefern

$f? : Datei$

$datei\_im\_verzeichnis! : \mathbb{B}$

$dat! : Datensatz$

$in\_verzeichnis\_lesen \mathrel{\widehat=}$

Rechteverwaltung

Modellieren Sie die Klasse *Rechteverwaltung*. Im einzelnen modellieren Sie bitte

1. das Zustandsschema, das eine geeignete Struktur namens *rechte* zur Speicherung der Rechte der einzelnen *Nutzer* auf *Datei*en enthält. *3 Punkte*

2. die Operationen *rechte_erweitern* und *rechte_einschränken*, mit denen die Rechte eines *Nutzer*s auf eine *Datei* eingeschränkt oder erweitert werden. Beide Operationen erhalten ein *Nutzer*- und ein *Datei*-Objekt. Vorraussetzung von *rechte_erweitern* ist, dass der *Nutzer* noch kein Zugriffsrecht für die *Datei* besitzt; *rechte_einschränken* darf nur angewandt werden, wenn der Nutzer das Zugriffsrecht besitzt. *4 Punkte*

3. die Operationen *hat_zugriff* und *hat_keinen_Zugriff*.
   hat_zugriff kann angewandt werden, wenn ein gegebener *Nutzer* eine gegebene *Datei* lesen darf. *hat_keinen_zugriff* lässt sich anwenden, wenn der *Nutzer* die übergebene *Datei* nicht lesen darf.
   Die Ausgabe *zugriff_ok!* ist genau dann wahr, wenn der *Nutzer* Zugriffsrechte auf eine *Datei* besitzt. *4 Punkte*

Rechteverwaltung

$\upharpoonright (rechte, rechte\_erweitern, rechte\_einschränken, hat\_zugriff, hat\_keinen\_zugriff)$

$rechte :$

rechte_erweitern

$\Delta(rechte)$
$user? : Nutzer$
$f? : Datei$

rechte_einschränken

$\Delta(rechte)$
$user? : Nutzer$
$f? : Datei$

hat_zugriff

$user? : Nutzer$
$f? : Datei$
$zugriff\_ok! : \mathbb{B}$

hat_keinen_zugriff

$user? : Nutzer$
$f? : Datei$
$zugriff\_ok! : \mathbb{B}$

Dateisystem

Dateisystem kombiniert die bisher modellierten Elemente zu einem sinnvollen Ganzen. Zur besseren Übersicht sind die Teile der Invariante der Klasse numeriert. *Sie müssen für diese Teilaufgabe nicht die gesamte Invariante verstehen.*

Modellieren Sie die folgenden Objekte und Operationen:

1. Definieren Sie an geeigneter Stelle in *Dateisystem* einen unveränderlichen *Nutzer admin*, der auf jede *Datei* im System zugreifen darf. Beschreiben Sie seine speziellen Rechte durch Invariante VII des *Dateisystem*s. *3 Punkte*
2. *datei_im_verzeichnis_lesen* erhält als Eingabe ein Verzeichnis und lässt in ihm die *Datei* suchen. *3 Punkte*
3. *suchen* soll prüfen, ob der übergebene *Pfad* im System enthalten ist, und das letzte *Verzeichnis* des übergebenen *Pfad*es ausgeben. *msg!* nimmt den Wert *ok!* an, wenn der *Datensatz* erfolgreich aus der *Datei* gelesen werden konnte. Sollte der *Nutzer* kein Recht auf die *Datei* besitzen, wird *msg!* mit *zugriff_nicht_gestattet* belegt. Sollte der Zugriff gestattet sein, der übergebene Pfad aber nicht gültig oder die gewünschte *Datei* nicht im letzten *Verzeichnis* des Pfades enthalten sein, gibt die Operation dementsprechende Fehlermeldungen(*datei_nicht_in_verzeichnis*, *pfad_nicht_im_system*) aus. *6 Punkte*
4. *lesen* ist eine Verknüpfung zwischen *suchen* und *datei_im_verzeichnis_lesen*, so dass die Operationen ihren Zweck erfüllen können. *3 Punkte*
5. *lesen_und_authorisieren* erhält als Eingabe einen *Nutzer*, einen *Pfad*, eine *Datei* und gibt eine *Nachricht* aus. Die Operation prüft, ob der *Nutzer* auf die angegebene *Datei* zugreifen darf. Wenn ja, leitet *lesen_und_authorisieren* die Anfrage an *lesen* weiter. *2 Punkte*

Invariante von Dateisystem

Untersuchen Sie nun die Invariante von *Dateisystem*

1. In der Invariante fehlt die Eigenschaft, dass auf jedem Pfad gilt: In der Sequenz, die den Pfad beschreibt, ist jeder direkte Nachfolger eines Elements der Sequenz in der *unterverzeichnisse*-Liste seines Vorgängers enthalten. Erweitern Sie die Invariante dahingehend unter VI.4. *3 Punkte*
2. Welche kritische Eigenschaft der Struktur des *Dateisystem*s wird durch Invariante VI.3 sichergestellt? (Ohne Beweis) *3 Punkte*

Dateisystem

$\upharpoonright (lesen\_und\_authorisieren, rechte\_erweitern, rechte\_einschränken)$

$wurzel : Verzeichnis$
$pfade : \mathbb{F}\, Path$
$rechteDB : Rechteverwaltung$
$nutzer : \mathbb{F}\, Nutzer$
Δ
$verzeichnisse : \mathbb{F}\, Verzeichnis$
$dateien : \mathbb{F}\, Datei$

(I) $verzeichnisse = \bigcup\{p : pfade \bullet ran(p)\}$

(II) $\forall\, d : verzeichnisse \bullet \forall\, e : d.unterverzeichnisse \bullet \exists\, p : pfade \bullet \exists\, i : dom(p) \bullet (e = p(i+1) \land d = p(i))$

(III) $\forall\, d1, d2 : verzeichnisse \mid d1 \neq d2 \bullet d1.dateien \cap d2.dateien = \varnothing$

(IV) $dateien = \bigcup\{d : verzeichnisse \bullet d.dateien\}$

(V) $rechteDB.rechte \subseteq nutzer \times dateien$

(VI) $\forall\, p : pfade \mid \#p > 1 \bullet$
(VI.1) $head(p) = wurzel \land$
(VI.2) $front(p) \in pfade \land$
(VI.3) $\neg\, \exists\, q : pfade \mid p \neq q \bullet last(p) = last(q) \land$
(VI.4)

(VII)

suchen

$p? : Pfad$
$zugriff\_ok? : \mathbb{B}$
$datei\_im\_verzeichnis? : \mathbb{B}$
$dir! : Verzeichnis$
$msg! : Nachricht$

$datei\_im\_verzeichnis\_lesen \mathrel{\widehat=}$

$lesen \mathrel{\widehat=}$

$lesen\_und\_authorisieren \mathrel{\widehat=}$

$rechte\_erweitern \mathrel{\widehat=} rechteDB.rechte\_erweitern$
$rechte\_einschränken \mathrel{\widehat=} rechteDB.rechte\_einschränken$