Модели на софтуерни системи

Проект

Тема 6: Регистър на наети превозни средства

Радослав Велков, МП СТ, 7MI3400582

A)

Предефинираните множества, които ще ползваме наготово са:

- Car - превозните средства от категория "Автомобили"
- Bike - превозните средства от категория "Мотопеди"
- Customer - клиентите, които наемат превозни средства

Всяко от тези множества се състои от хомогенни елементи. Всеки един от тези елементи съдържа характеристики, правещи го уникален за базата данни на системата, които обаче не са предмет на нивото на абстракция, на което ще опишем софтуерната системата чрез Z-Schema, и съответно няма да бъдат включени.

VehicleRental
rentedCars : Car ⇸ Customer rentedBikes : Bike ⇸ Customer rentedCarsDays : Car ⇸ ℕ rentedBikesDays : Bike ⇸ ℕ countRentedVehicles: Customer → ℕ cars : ℙ Car bikes : ℙ Bike
cars ⊇ dom(rentedCars) bikes ⊇ dom(rentedBikes)

Основната схема на системата VehicleRental. Тя се явява нейното начално състояние и съдържа:

- Декларации:
  - rentedCars - частична функция съпоставяща на всеки нает автомобил най-много един наемател
  - rentedBikes - частична функция съпоставяща на всеки нает мотопед най-много един наемател
  - rentedCarsDays - частична функция съпоставяща на всеки нает автомобил естествено число - дните за които е нает
  - rentedBikesDays - частична функция съпоставяща на всеки нает мотопед естествено число - дните за които е нает
  - countRentedVehicles - тотална функция съпоставяща на всеки клиент естествено число - брой наети ПС (автомобили и/или мотопеди)
  - cars - множеството от всички автомобили, които системата поддържа. Явява се подмножество на множеството от всички коли - Car
  - bikes - множеството от всички мотопеди, които системата поддържа. Явява се подмножество на множеството от всички мотопеди - Bike

За cars и bikes сме длъжни да дадем най-голямото възможно покриващо множество и затова ползваме ℙ (Power Set).

- Предикати:
  - cars със сигурност трябва да е надмножество на домейна на rentedCars
  - bikes със сигурност трябва да е надмножество на домейна на rentedBikes

Б)

Дефинираме нова променлива от свободен тип *Response*:

Response ::= rented | not-rented

Операция по наемане на конкретна кола за точен брой дни:

RentACar

ΔVehicleRental car? : Car days? : ℕ customer? : Customer response! : Response
(car? ∉ dom(rentedCars) ∧ countRentedVehicles(customer?) < 3 ∧ rentedCars′ = rentedCars ∪ {car? ↦ customer?} ∧ rentedCarsDays′ = rentedCarsDays ∪ {car? ↦ days?} ∧ countRentedVehicles′ = countRentedVehicles ⊕ {customer? ↦ (countRentedVehicles(customer?) + 1)} ∧ response! = rented) ∨ ((car? ∈ dom(rentedCars) ∨ countRentedVehicles(customer?) ≥ 3) ∧ rentedCars′ = rentedCars ∧ rentedCarsDays′ = rentedCarsDays ∧ countRentedVehicles′ = countRentedVehicles ∧ response! = not-rented)

Като вход за тази операция са нужни: конкретна кола, за колко дни се иска да бъде наета и кой е клиентът, който ще я наема. Операцията има потенциал за модифициране на системата ΔVehicleRental. В резултат връща отговор за успеха на операцията.

В положителния случай, когато желаният автомобил не е нает и клиентът не е достигнал лимита си от 3 ПС тогава операцията е успешна, което добавя: нова връзка между кола и клиент към rentedCars, нова връзка между кола и брой дни, за които е наета в rentedCarsDays, както и инкрементира броя на наетите ПС за дадения клиент в countRentedVehicles чрез "Relational Overriding" оператора ⊕. Също така, връща отговор, че ПС е нает (rented).

В обратния случай, когато автомобилът е вече нает ИЛИ клиентът е достигнал лимита от 3 ПС, не се случват промени и се връща отговор, че не е извършено наемане (not-rented).

Аналогична операция по наемане на конкретен мотопед за точен брой дни:

RentABike

ΔVehicleRental
bike? : Bike
days? : $\mathbb{N}$
customer? : Customer
response! : Response

(bike? $\notin$ dom(rentedBikes) $\land$
countRentedVehicles(customer?) < 3 $\land$
rentedBikes′ = rentedBikes $\cup$ {bike? $\mapsto$ customer?} $\land$
rentedBikesDays′ = rentedBikesDays $\cup$ {bike? $\mapsto$ days?} $\land$
countRentedVehicles′ =
countRentedVehicles $\oplus$ {customer? $\mapsto$ (countRentedVehicles(customer?) + 1)} $\land$
response! = rented)
$\lor$
((bike? $\in$ dom(rentedBikes) $\lor$ countRentedVehicles(customer?) $\geq$ 3) $\land$
rentedBikes′ = rentedBikes $\land$
rentedBikesDays′ = rentedBikesDays $\land$
countRentedVehicles′ = countRentedVehicles $\land$
response! = not-rented)

За пълнота на системата, дефинираме и генерализирана схема за операция по наемане на превозно средство.

RentAVehicle

RentACar
RentABike

или иначе казано RentAVehicle $\triangleq$ RentACar $\land$ RentABike. Извършва се обединение на декларациите на двете схеми и на предикатите на двете схеми.

Алтернатива е вместо RentACar, която съдържа както успешно наемане на кола, така и неуспешно, да дефинираме SuccessfulRentACar, която само модифицира системата (ΔVehicleRental) и отделно да дефинираме NonSuccessfulRentACar, която няма да я модифицира (ΞVehicleRental), като и двете не се ангажират с върнат резултат.

Отделно дефинираме схеми за положителен и отрицателен резултат:

Success

response! : Response

---

response! = rented

и

Failure

response! : Response

---

response! = not-rented

И тогава ще имаме обща схема RentACar, която ще представлява:

RentACar $\triangleq$ (SuccessfulRentACar $\land$ Success) $\lor$ (NonSuccessfulRentACar $\land$ Failure).

Аналогично може да се направи и за RentABike.

Операция по връщане на нает автомобил (описана по-просто без връщане на резултат):

ReturnACar

ΔVehicleRental car? : Car customer? : Customer
car? ∈ dom(rentedCars) ∧ rentedCars′ = rentedCars \ {car? ↦ customer?} ∧ rentedCarsDays′ = rentedCarsDays \ {car? ↦ days?} ∧ countRentedVehicles′ =   countRentedVehicles ⊕ {customer? ↦ (countRentedVehicles(customer?) - 1)}

Когато се връща наета кола от даден клиент, тя се премахва от функциите rentedCars и rentedCarsDays и стойността съпоставена на клиента в countRentedVehicles се декрементира с 1.

Аналогична е операцията по връщане на нает мотопед от клиент:

ReturnABike

ΔVehicleRental bike? : Bike customer? : Customer
bike? ∈ dom(rentedBikes) ∧ rentedBikes′ = rentedBikes \ {bike? ↦ customer?} ∧ rentedBikesDays′ = rentedBikesDays \ {bike? ↦ days?} ∧ countRentedVehicles′ =   countRentedVehicles ⊕ {customer? ↦ (countRentedVehicles(customer?) - 1)}

B)

Схема на проста справка на всички клиенти наели автомобил:

ReportAllCustomersThatRentedCars

ΞVehicleRental customersCars! : ℙ Customer
customersCars! = ran(rentedCars)

customersCars е върнатият резултат - множество съдържащо всички клиенти, които са наели кола в момента. Това всъщност представлява точно образа (range) на функцията rentedCars.

По-сложна справка удовлетворяваща условието би могла да бъде:

ReportAllCustomersThatRentedCars

ΞVehicleRental customersCars! : ℙ (Car × Customer × ℕ)
customersCars! =   {(c,cust,d) | c ∈ dom(rentedCars) ∧ rentedCars(c) = cust ∧ rentedCarsDays(c) = d}

Тук върнатият резултат е множеството от всички наредени тройки (кола,клиент,дни), като така отразяваме, че дадена кола е наета от даден клиент за точно определен брой дни. c ∈ dom(rentedCars) - колата c е в дефиниционното множество (домейна) на функцията rentedCars, тоест е наета. Колата е наета от клиента cust, където cust ∈ Customer = ran(rentedCars) и е наета за d ∈ ℕ = ran(rentedCarsDays) дни.

Аналогичните схеми се дефинират и за мотопеди:

ReportAllCustomersThatRentedBikes

ΞVehicleRental
customersBikes! : ℙ Customer

customersBikes! = ran(rentedBikes)

ReportAllCustomersThatRentedBikes

ΞVehicleRental
customersBikes! : ℙ (Bike × Customer × ℕ)

customersBikes! =
{(b,cust,d) | b∈dom(rentedBikes) ∧ rentedBikes(b) = cust ∧ rentedBikesDays(b) = d}

Г)

Схема на функция за изчисляване на броя на колите, които са наети за поне даден брой дни:

RentedCarsPerDays

ΞVehicleRental
days? : ℕ
carsCount! : ℕ

carsCount! = #{c : dom(rentedCarsDays) | rentedCarsDays(c) ≥ days?}

използва само функцията rentedCarsDays от VehicleRental като върнатият резулат е цяло число - броя на елементите от множеството състоящо се от всяка стойност от домейна на rentedCarsDays (т.е. всеки нает автомобил), но филтрирани по това, че се взимат само тези, които са наети за не по-малко от въведния брой дни като вход на функцията.

Онагледено с пример:

Ако rentedCarsDays = {Fiat ↦ 9, VW↦ 7, BMW↦ 3, Audi ↦ 5}, то при:

- days? = 5 ще имаме carsCount! = 3 (което са Fiat, VW, Audi);
- days? = 6 ще имаме carsCount! = 2 (което са Fiat, VW);
- days? = 8 ще имаме carsCount! = 1 (което е само Fiat).

Специфицираме и аналогична функция за мотори:

RentedCarsPerDays

ΞVehicleRental
days? : ℕ
bikesCount! : ℕ

bikesCount! = #{b : dom(rentedBikesDays) | rentedBikesDays(b) ≥ days?}