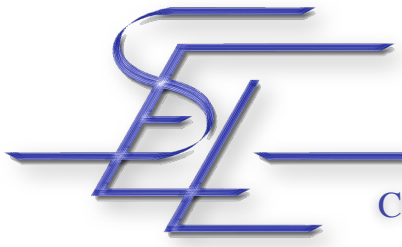
Solutions Midterm 2010

Jörg Kienzle

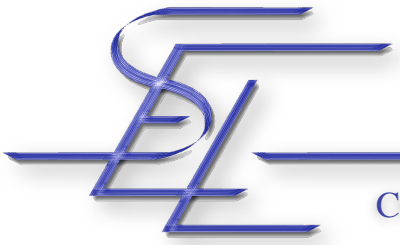


McGill

Car Insurance Domain Model Question (1)

Problem Description

An insurance company wants to automate its functions. The company sells several kinds of insurance policies, including life insurance and car insurance, and has plans to expand its services. Customers can purchase insurance policies, pay insurance fees accordingly, and receive compensation. Each insurance purchase activity is overseen by a sales man and results in a contract, in which the form of payment (e.g. yearly, monthly, ...) is specified. Claim agents are in charge of handling claims. They study the claims received against the insurance plans and decide on how much compensation the client is entitled to. All claims, even those that are not compensated, have to be archived in case of future reclamations.



McGill

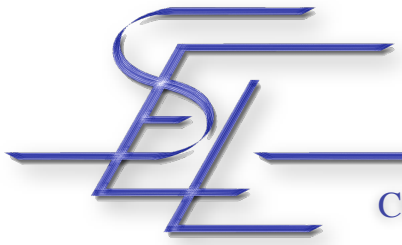
Car Insurance Domain Model Question (2)

Life insurance policies

When purchasing a life insurance, the customer has to nominate at least one beneficiary (who will receive the compensation in case of death of the customer). For life insurance policies, the yearly fee is based on the persons age and the date the contract was signed.

Car insurance policies

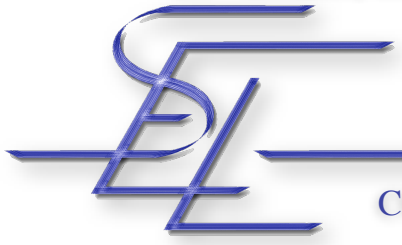
The fee for a car insurance is based on the price of the car and the year in which the car was built. The base car insurance covers accidents for the primary driver only (i.e. the customer himself/herself). It is however possible to add a maximum of 5 secondary drivers to the contract. Each secondary driver costs $10 extra per year. This total (base fee + secondary drivers) is then further adjusted based on the claim history. For each accident that has been compensated in the past, the fee is increased by 20%.



McGill

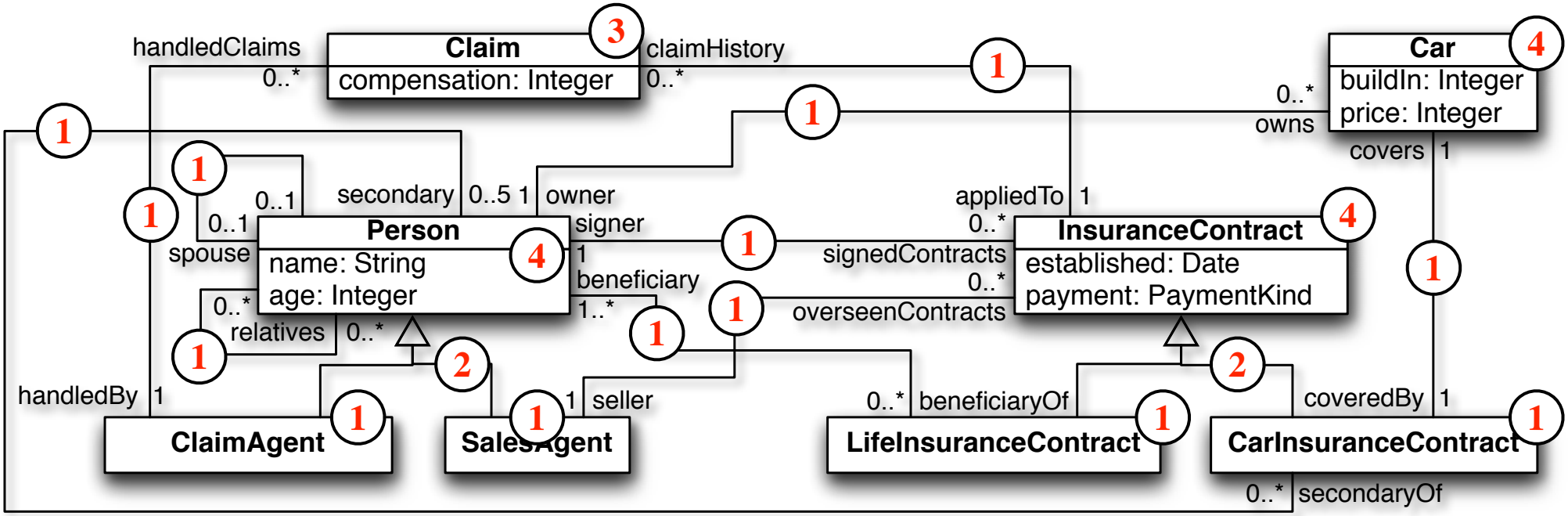
Car Insurance Domain Model Question (3)

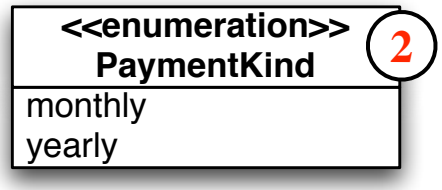
1. Devise a domain model that models the concepts of a (single) insurance company. Before you start, read also task 2 and the description of the OCL constraints that you'll have to write in task 3. They might require additional concepts / associations / attributes that you need to add to your domain model.
2. Explain how, if ever the company decides to add new insurance policies (like a house insurance) to their portfolio, your model would have to be updated.
3. Write the following constraints and functions in OCL (if your model already models that constraint, then just write: "Is covered by model")
   1. Every car has to be covered by exactly one insurance policy.
   2. A car insurance has to be bought by the owner of the car.
   3. A customer's spouse has to be declared as a secondary driver.
   4. Write an OCL function that, given a base fee, calculates the adjusted fee for a car insurance policy.
   5. Sales agents and claim agents should not be handling policies of their relatives, or policies in which they are involved as beneficiaries or secondary drivers. (You can either write one invariant, or multiple invariants to answer this question.)



McGill

Insurance Company Domain Model





context p: Person:
inv: p.relatives→**includesAll**(p.spouse)

35 points total + 5 bonus

New insurance policies can be added by adding new subclasses to the Insurance Contract Class

Insurance Domain Model OCL

Write the following OCL constraints:

1. Every car has to be covered by exactly one insurance policy.

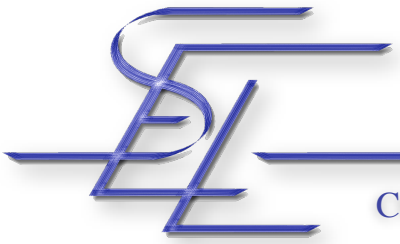
① **Covered by model**

2. A car insurance has to be bought by the owner of the car.

② **context** c: CarInsuranceContract
inv: c.covers.owner = c.signer

3. A customer's spouse has to be declared as a secondary driver.

③ **context** c: CarInsuranceContract
inv: c.signer.spouse→**notEmpty() implies**
c.secondary→**includes**(c.signer.spouse)



McGill

Insurance Domain Model OCL

4. Write an OCL function that, given a base fee, calculates the adjusted fee for a car insurance policy.

(4)

context c: CarInsuranceContract
def: adjustedFee(baseFee: Integer) : Integer =
(baseFee + c.secondary→**size**() * 10) *
(1 + c.claimHistory→**select**(compensation > 0)→**size**() * 0.2)

5. Sales agents and claim agents should not be handling policies of their relatives, or policies in which they are involved as beneficiaries or secondary drivers.

(5)

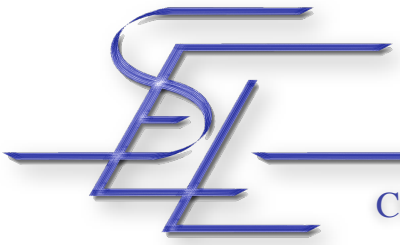
context s: SalesAgent:

inv: s.overseenContracts.signer→**excludesAll**(s.relatives→**including**(s)) **and**
s.overseenContracts→**excludesAll**(s.beneficiaryOf→**union**(s.secondaryOf))

context c: ClaimAgent:

inv : c.handledClaims.appliedTo.signer→**excludesAll**(s.relatives→**including**(s)) **and**
c.handledClaims.appliedTo→**excludesAll**(c. beneficiaryOf→**union**(c.secondaryOf))

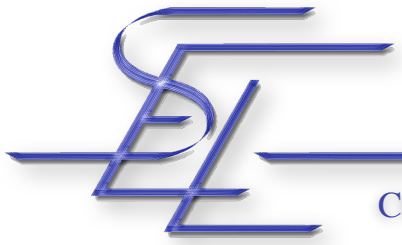
15 points total



McGill

Recycling Machine Use Case

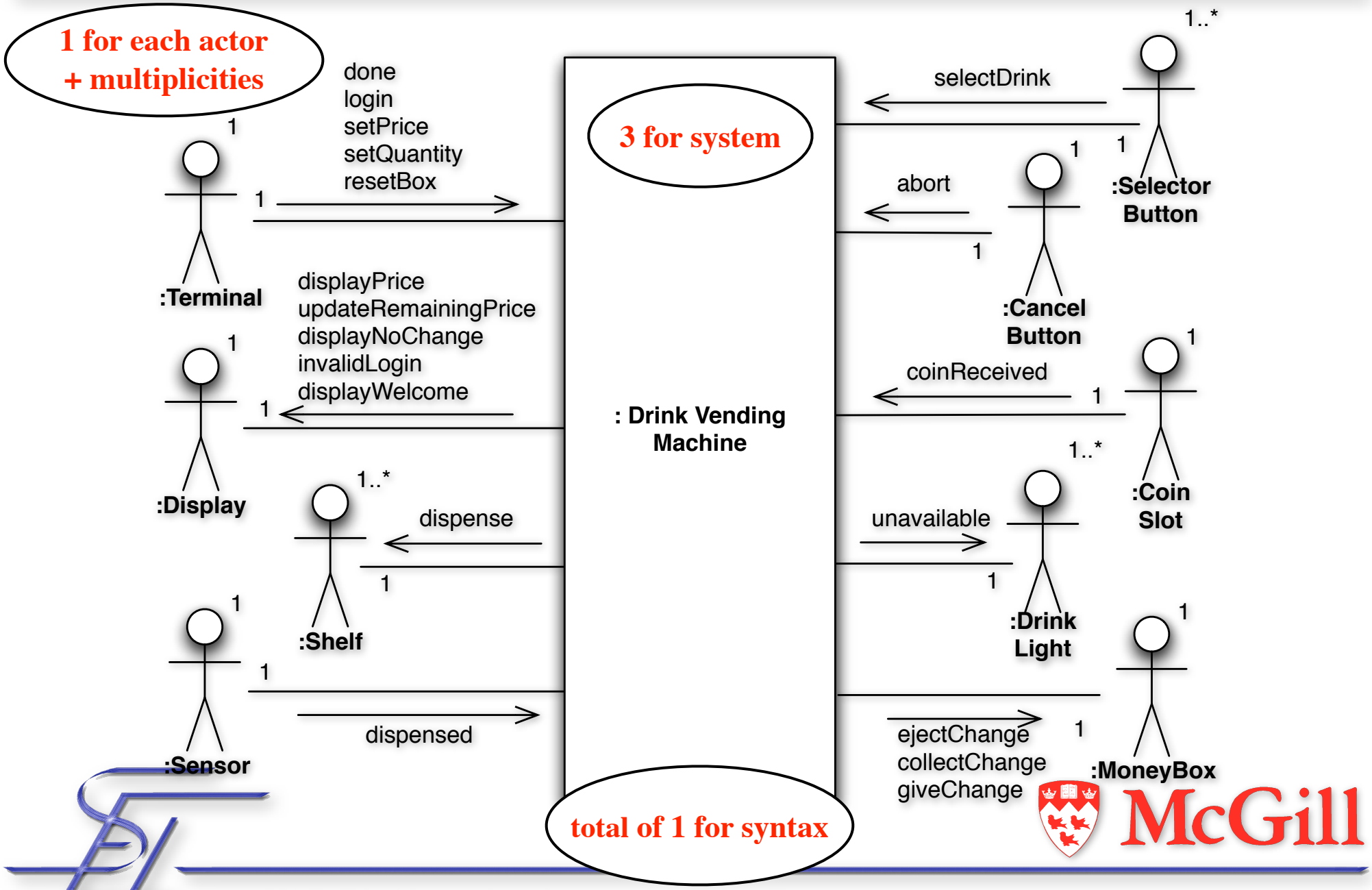
1 **Use Case**: RecycleItems
1 **Scope**: RecyclingMachine
1 **Level**: User-Goal
1 **Intention in Context**: The *User* wants to recycle bottles and cans in exchange for money.
1 **Multiplicity**: Only one *User* can recycle items at a given time.
1 **Primary Actor**: User
3 **Secondary Actors**: Recognizer, Display, (FinishedButton), Printer
Main Success Scenario:
1 *Step 1, 2 and 3 are repeated for each item the User wishes to recycle.*
1. *User inserts a can or a bottle into the recognizer.*
2 2. *Recognizer* informs *System* about the item that was recognized.
2 3. *System* acknowledges recognition to User by updating the refund total on the *Display*.
2 4. *User* informs *System* that s/he has no more items to process (using the *FinishedButton*).
2 5. *System* prints receipt using *Printer*.
Extensions:
2 2a. System rejects the inserted item (because it does not recognize it, or because the store does not accept that kind of item). Use case continues with next item at step 1.



20 points total

McGill

DrinkVendingMachine Environment Model

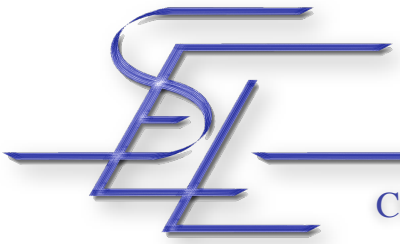


Drink Vending Messages (1)

- Input (grouped by use case)

① selectDrink (no parameter needed since **sender** encodes drink)
① dispensed
① coinReceived(c: Coin)
① abort

① login(username: String, password: EncryptedString)
① setPrice(d: Drink, amount: Integer)
① setQuantity(d: Drink, quantity: Integer)
① resetBox (no parameter needed, or else change amounts)
① done



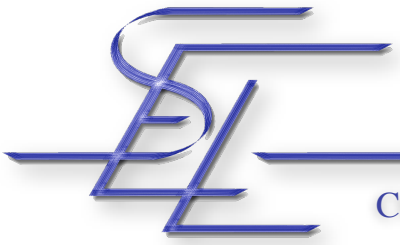
Drink Vending Messages (2)

- Output (grouped by use case)

(1) displayPrice(amount: Integer)
(updateRemainingPrice(amount: Integer))
(1) dispenseDrink
(1) displayNoChange
(1) ejectChange
(1) giveChange(amount: Integer)
(collectChange)
(1) unavailable

(1) displayWelcome (or acknowledgment)
(1) invalidLogin

30 points total



McGill