**TERM PROJECT**

Akhilesh Kumar

CSE 621

Instructor: Dr. Ann E. Sobel

**DEFINITIONS FOR THE ROYAL SERVICE SYSTEM:**

UNITPRICE : ℝ (Real Numbers)

PARKTYPE : **daily | weekly | monthly**

PAID :  **true | false**

THRESHOLD : ℕ1

DATE : DAY : MONTH : YEAR

DAY : [1-31]

MONTH : [1-12]

YEAR : [1-3000]

PRICE : ℝ

ACCEPTANCE :  **true | false**

PAYMENTMODE : **instant | monthly**

PAYMENTBY **: cash | creditcard | check | later**

USERID : ℕ

PRODUCTID : ℕ

LASTACTIVITYDATE : DATE

CREDITSCORE : ℕ

STRING : [a-z]\*

Note: This model uses X as the multiplication operator as opposed to the middle dot which was difficult to differentiate from the dot (such that).

**The Z- SCHEMA FOR THE SYSTEM**

The model is shown below and the description of the model follows the model. The explanation of the model is provided below it.

\_\_\_\_RoyalServiceStation\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

users : USERID I---> LASTACTIVITYDATE

products : PRODUCTID I---> UNITPRICE

servicerecords : USERID x SERVICEID x DATE I---> PRICE

payment : SERVICEID x PAID x PAYMENTBY I---> PAYMENTMODE

inventoryrecords : PRODUCTID x THRESHOLD I---> QUANTITY

park : PARKTYPE I---> PRICE

availablespace : ℕ

creditcardinfo : USERID I---> CREDITSCORE

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

∀ i, j . 1 ≤ i, j ≤ |users| . i ≠ j => dom1 users i ≠ dom1 users j

∀ i, j . 1 ≤ i, j ≤ |park| . i ≠ j => dom1 park i ≠ dom1 park j

∀ i, j . 1 ≤ i, j ≤ |products| . i ≠ j => dom1 products i ≠ dom1 products j

∀ i, j . 1 ≤ i, j ≤ |servicerecords| . i ≠ j => dom2 servicerecords i ≠ dom2 sevicerecords j

∀ i, j . 1 ≤ i, j ≤ |inventoryrecords| . i ≠ j => dom1 inventoryrecords i ≠ dom1 inventoryrecords j

∀ i, j . 1 ≤ i, j ≤ |payment| . i ≠ j => dom1 payment i ≠ dom1 paayment j

∀ i, j . 1 ≤ i, j ≤ |creditcardinfo| . i ≠ j => dom1 creditcardinfo i ≠ dom1 creditcardinfo j

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The model for the Royal Service Station is shown above. The ‘users’ is a function from the USERID to the LASTACTIVITY day. The LASTACTIVITYDAY represents the day when the user last used the services of the system. The ‘products’ is a function from the product id to the unit price associated with the product. The ‘servicerecords’ has been used to maintain the records of each service i.e. who used the service, when it was used and the cost of the service. The ‘payment’ keeps track of how the payment was made for each service i.e. paid in cash or by check or by credit card, whether the payment was made instantly or through the monthly bill. The ‘inventoryrecords’ keeps the track of each product. The minimum quantity the system might have and the actual quantity of the product in the inventory. The ‘park’ is a function from the type of parking(daily or weekly or monthly) and the price associated with it. The ‘availablespace’ is used to keep track of the parking spaces in the parking lot. The ‘creditcardinfo’ has the information about the user and his credit score. The predicate part of the model says that the USERID in the ‘users’ and ‘creditcardinfo’, PRODUCID in the ‘products’ and ‘inventoryrecords’ SERVICEID in the ‘servicerecords’ and ‘payment’ should be unique. These domain values might not be duplicated.

**THE ∆ SCHEMA FOR THE MODEL**

The potential changes of the system can be represented by the schema below. The products and the park functions are mapping from the product id or the parking type to the price associated. So they remain unaffected.

\_\_\_\_\_\_\_\_\_∆RoyalServiceStation\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

RoyalServiceStation

users’ : USERID I---> LASTACTIVITYDATE

products’ : PRODUCTID I---> UNITPRICE

servicerecords’ : USERID x SERVICEID x DATE I---> QUANTITY

payment’ : SERVICEID x PAID I---> PAYMENTMODE

inventoryrecords’ : PRODUCTID x THRESHOLD I---> QUANTITY

park’ : PARKTYPE I---> PRICE

availablespace’ : ℕ

creditcardinfo’ : USERID I---> CREDITSCORE

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

products’ = products

park’ = park

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**THE 𝚵 SCHEMA FOR THE MODEL**

This schema will be used by the operations when the system does not change.

\_\_\_\_\_\_\_\_\_𝚵RoyalServiceStation\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

∆ RoyalServiceStation

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

users’ = users

products’ = products

servicerecords’ = servicerecords

payment’ = payment

inventoryrecords’ = inventoryrecords

park’ = park

availablespace’ = availablespace

creditcardinfo’ = creditcardinfo

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**VARIOUS SYSTEM FUNCTIONS ARE SHOWN BELOW**

**ADDING FUEL TO TANK**

Adding fuel to the tank is one of the three major services that are provided by the system. The user should have record in the system to use the services. When he adds fuel the system returns the cost and updates the last activity date of the user to the current date. The inventory needs to be updated by the amount of fuel the user used. The system also keeps track of how the user paid. The system should also have the amount of gas requested by the user(this needs to be verified in the predicate section). The service record should be updated as shown in the schema below. If the user decides to pay on monthly bill then the information is stored as not paid otherwise it is set to paid and also his credit score should not be zero. The user can pay by cash, credit card, check or later.

\_\_\_\_\_\_\_\_\_\_\_\_Refuel\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

∆RoyalServiceStation

user? : USERID

gas? : PRODUCTID

serviceno? : SERVICEID

date? : LASTACTIVITYDATE

amount? : QUANTITY

paymenttype? : PAYMENTMODE

paymethod? : PAYMENTBY

cost! : PRICE

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

user? ∈ dom (users )

gas? ∈ dom (products)

users’ = users ⊕ { user? I---> date? }

∃ i . 1 ≤ i ≤ |inventoryrecords| . ( dom1 inventoryrecords i = gas? ) => amount? > inventoryrecords i

cost! = amount? X products(gas?)

servicerecords’ = servicerecords U { user? x serviceno? x date? I---> cost! }

∃ i . 1 ≤ i ≤ |inventoryrecords| . ( dom1 inventoryrecords i = gas? ) => inventoryrecords’ = inventoryrecords ⊕ { gas? x dom2 inventoryrecords i I---> (inventoryrecords i) – amount? }

(paymenttype? = **instant**) => payment’ = payment U {serviceno? x **true** x paymethod?I---> paymenttype? }

∃ i . 1 ≤ i ≤ |creditcardinfo| . (user? = dom1 creditcardinfo i) . ((paymenttype? = **monthly)** ^( creditcardinfo i ≠ **0**)) => payment’ = payment U { serviceno? x **false** x paymethod? |---> paymenttype? }

availablespace’ = availablespace

creditcardinfo’ = creditcardinfo

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**VEHICLE MAINTENANCE SERVICES**

Explanation is provided below the schema.

\_\_\_\_\_\_\_\_\_\_\_\_VehicleMaintenance\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

∆RoyalServiceStation

user? : USERID

part? : PRODUCTID

serviceno? : SERVICEID

laborcost? : PRICE

date? : LASTACTIVITYDATE

quantity? : QUANTITY

paymethod? : PAYMENTBY

paymenttype? : PAYMENTMODE

cost! : PRICE

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

user? ∈ dom ( users)

part? ∈ dom (products)

users’ = users ⊕ { user? I---> date? }

cost! = laborcost? +( products(part?) X quantity? )

servicerecords’ = servicerecords U { user? x serviceno? x date? I---> cost! }

∃ i . (dom1 inventoryrecords i = part?) => (quantity? > inventoryrecords i)

∃ i . 1 ≤ i ≤ |inventoryrecords| . (dom1 inventoryrecords i = part?) => inventoryrecord’ = inventoryrecords ⊕ { part? x dom2 inventoryrecords i I---> (inventoryrecords i) – quantity? }

(paymenttype? = **instant**) => payment’ = payment U {serviceno? x **true** x paymethod? I---> paymenttype? }

∃ i . 1 ≤ i ≤ |creditcardinfo| . (user? = dom1 creditcardinfo i) . ((paymenttype? = **monthly)** ^ ( creditcardinfo i ≠ **0**)) => payment’ = payment U { serviceno? x **false** x paymethod? |---> paymenttype? }

availablespace’ = availablespace

creditcardinfo’ = creditcardinfo

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The second major service provided by the system is Vehicle maintenance. The schema output price after the service. The user information is updated to store the new activity date. The system should update the inventory after selling the parts to customer. In this case the inventory record associated with the product id of the part is lowered by the amount used by the user. The system should also check that it has required amount of products ordered by the customer. The service records should be updated for the services provided to the customer. The payment information is updated depending on how the user decides to pay(just like the previous schema). If he pays instantly then the information is stored as paid otherwise it is paid later in monthly bills. In case of payment by monthly bill the schema checks that his credit score is not zero.

**PARKING SERVICES**

The system checks whether the parking space is available. The system also outputs the cost associated with the service provided. The payment information should be updated for the system based upon the payment type which can be monthly bill or an instant payment. The service records is also updated. If the parking space is available then the user can use the parking services and the available parking space for the system is decreased. The user information is updated changing the last activity date to the new date on which the service is used. If the user asks for monthly bill then the payment information is set to not paid otherwise if the user pays it instantly then it is set to paid. In case his credit score is zero then he cannot order by monthly bill.

\_\_\_\_\_\_\_\_\_\_\_\_ParkingService\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

∆RoyalServiceStation

user? : USERID

unittime? : ℕ

serviceno? : SERVICEID

parkingtype? : PARKTYPE

paymethod? : PAYMENTBY

date? : LASTACTIVITYDATE

amount? : QUANTITY

paymenttype? : PAYMENTMODE

cost! : PRICE

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

user? ∈ dom (users)

parkingtype? ∈ dom (park)

users’ = users ⊕ { user? I---> date? }

cost! = park(parkingtype?) X unittime?

servicerecords’ = servicerecords U { user? x serviceno? x date? I---> cost! }

inventoryrecords’ = inventoryrecords

(paymenttype? = **instant**) => payment’ = payment U {serviceno? x **true** x paymethod?I---> paymenttype? }

∃ i . (1 ≤ i ≤ |creditcardinfo|) . (user? = dom1 creditcardinfo i) . ((paymenttype? = **monthly)** ^( creditcardinfo i ≠ **0**)) => payment’ = payment U { serviceno? x **false** x paymethod?---> paymenttype? }

(availablespace’ = availablespace – 1) ^ (availablespace’ >= 0)

creditcardinfo’ = creditcardinfo

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**DISCOUNT BY MANAGER**

The manager is capable to give discounts to any user for any of the service he takes advantage of. The system should check that the discount given to the user should not be more than the price of the service. The price associated with the service should be modified accordingly. Here in this schema we only need to modify the price information of the service for which the Manager is giving the discount to the user. Since the price information is only in the ‘servicerecords’ we need to modify only that information. All other information remains unaffected in the system.

\_\_\_\_\_\_\_\_\_\_\_\_ManagersDiscount\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

∆RoyalServiceStation

user? : USERID

serviceno? : SERVICEID

discount? : PRICE

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

user? ∈ dom (users)

serviceno? ∈ dom2 (servicerecords)

users’ = users

∃i . 1 ≤ i ≤ |servicerecords| . (serviceno? = dom2 servicerecords i) => discount? ≤ servicerecords i

payment’ = payment

availablespace’ = availablespace

inventoryrecords’ = inventoryrecords

creditcardinfo’ = creditcardinfo

∃ i . ( 1 ≤ i ≤ |servicerecords| ) . ( dom2 servicerecords i = serviceno? ) => servicerecords’ = servicerecords ⊕ { user? x serviceno? x dom3 servicerecords i I---> (servicerecords i - discount?) }

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**SALES TAX**

5% extra local sales tax is applicable on all the purchases. The system displays the tax for each purchase and adds the sales tax to the cost of the service. The service records must only be updated as that keeps track of the price associated with each service. The service record information is updated to include the local sales tax on all purchases.

\_\_\_\_\_\_\_\_\_\_\_\_LocalSalesTax\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

∆RoyalServiceStation

serviceno? : SERVICEID

tax! : PRICE

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

serviceno? ∈ dom2 (servicerecords)

users’ = users

inventoryrecords’ = inventoryrecords

payment’ = payment

∃ i . ( 1 ≤ i ≤ |servicerecords| . (serviceno? = dom2 servicerecords i ) => ( servicerecords’ = servicerecords ⊕ { dom1 servicerecords i x serviceno? x dom3 servicerecords i I---> ( servicerecords i X 1.05) ) ^ (tax! = servicerecords i X **0.05)**

availablespace’ = availablespace

creditcardinfo’ = creditcardinfo

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**TRACKING MONTHLY BILL**

Monthly Bill for any user can be traced out of the system. All the services used the user and the price associated with those services is output. The schema looks for all the services used between the dates provided and checks if the service was availed by the user for which the monthly bill is being tracked then that service information is displayed. In fact the schema is capable of tracking bill for any user between any period of time.

\_\_\_\_\_\_\_\_\_\_\_TrackMonthlyBill\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

𝚵 RoyalServiceStation

user? : USERID

serviceno! : SERVICEID

price! : PRICE

firstmonthday? : DATE

lastmonthday? : DATE

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

user? ∈ dom (users)

∀ i . ( (1 ≤ i ≤ |servicerecords| . (dom1 servicerecords i = user?) ^ ( firstmonthday? < dom3 servicerecords i < lastmonthday? ) => (serviceno ! = dom2 servicerecords i ) ^ ( price! = servicerecords i) )

firstmonthday? < lastmonthday?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**DAY TO DAY RECORDS**

The day to day records can be obtained by the system if needed. The operation can be represented by the schema shown below. The date is input for the schema below and all the services provided by the system on that day is output of the schema. The cost associated with each service also forms the part of output so that proper day to day analysis can be made. The system does not change because the services of the system are not being.

\_\_\_\_\_\_\_\_\_\_\_DayToDayRecords\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

𝚵 RoyalServiceStation

date? : DATE

user! : USERID

serviceno! : SERVICEID

price! : PRICE

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

∀ i . ( (1 ≤ i ≤ |servicerecords| . (date? = dom3 servicerecords i ) => ( user! = dom1 servicerecords i) ^ ( service! = dom2 servicerecords i) ^ ( price! = servicerecords i))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**LOW INVENTORY**

The system is capable of giving warning about the low inventory. Each product has some threshold value. If the product goes below that threshold value the system warns about the low inventory. The schema checks the inventory for all the products and checks whether the quantity is less than the threshold value for that product, then it displays those products which have available quantity less than the required.

\_\_\_\_\_\_\_\_\_\_\_LowInventory\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

𝚵 RoyalServiceStation

Id! : PRODUCTID

----------------------------------------------

∀ i . ( (1 ≤ i ≤ |inventoryrecords|) . (inventoryrecords i < dom2 inventoryrecords i) => (id! = dom1 inventoryrecords i))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**AVAILABLE PARKING SPACES**

The system tells the manager about the available parking spaces on demand. The operation shown below gives the parking spaces that are not occupied.

\_\_\_\_\_\_\_\_\_\_\_DisplayAvailablespace\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

𝚵 RoyalServiceStation

available! : ℕ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

available! = availablespace

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**DORMANT ACCOUNTS**

The users who do not make use of any services in the last two month are notified of their dormant accounts. The system here outputs the dormant user id so that they can be sent appropriate messages. The schema checks all the users who do not have any activity in past two months and then sends them a message using DormancyMessage helper method which takes the user information as input.

\_\_\_\_\_\_\_\_\_\_\_DormantAccount\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

𝚵 RoyalServiceStation

currentdate? : DATE

dormantusers! : USERID

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

∀ i . ( (1 ≤ i ≤ |users| . (currentdate? – users i) > **60** => (dormantusers! = dom users i) ^ DormancyMessage(dormantuser!))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**PERIODIC MESSAGES**

A periodic message in six months is sent to the users to remind them of Vehicle Maintenance. The CheckDate() helper function takes current date as input and outputs true it is the first day of January or June, otherwise it returns false. It will then send periodic message to all the users about the vehicle maintenance. SendPeriodicMessage is a helper function that sends message to all users.

\_\_\_\_\_\_\_\_\_\_\_PeriodicMessage\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

𝚵 RoyalServiceStation

currentdate? : DATE

result! : ACCEPTANCE

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

result! = CheckDate(currentdate?)

(result! = **true )** => SendPeriodicMessage()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**PAYMENT OVERDUE**

Any user with payment overdue is sent message. If any bill that is more than 2 months old is unpaid then the user is sent a warning message. There will be no changes in the system as the services of the system are not being used. All the information about the services taken from the system is scanned and the services that are more than two months old are still unpaid are checked and the user information and the price of the service is output so that message is sent. SendMessage is a helper function that takes user, service and the price information and sends warning messages to the user telling about the service and the price information of the service used by them.

\_\_\_\_\_\_\_\_\_\_\_\_WarnUsers\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

𝚵RoyalServiceStation

currentdate? : DATE

price! : PRICE

user! : USERID

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

∀ i ∃j . ( 1 ≤ i ≤ |servicerecords| . 1 ≤ j ≤ |payment|. ( dom2 servicerecords i = dom1 payment j) ^ ( dom2 payment j= **false**)^ ((currentdate? – dom3 servicerecords i) > **60**) => (user! = dom1 servicerecords i ) ^ (price! = servicerecords i) ^ SendMessage(user!, dom2 servicerecords i, price!)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CANCEL CREDIT**

Any service whose payment is not made within 3 months leads to cancellation of the user’s credit. The system checks all the services for which the payment is not made and is over three months of time then the credit of the user is cancelled. If the credit of a user is cancelled then his ‘creditcardifo’ which stores the information about the user’s credit only needs to be changed. The user, product, inventory, payment, servicerecords remain unchanged. The last predicate in the schema says that the for every service there is a payment record and the user information of that service, the schema checks which service is over three months old and the payment for that service is still unpaid; if such condition occurs then the system sets the credit score for that user to 0. The other things in the system should be unaffected as they have nothing to do with the credit information of any user.

\_\_\_\_\_\_\_\_\_\_\_\_CancelCredit\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

∆RoyalServiceStation

currentdate? : DATE

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

users’ = users

inventoryrecords’ = inventoryrecords

payment’ = payment

payment’ = payment

servicerecords’ = servicerecords

availablespace’ = availablespace

∀ i ∃j ∃k . ( 1 ≤ i ≤ |servicerecords| . 1 ≤ j ≤ |payment| . 1 ≤ k ≤ |creditcardinfo| . ( dom2 servicerecords i = dom1 payment j) ^ ( dom2 payment j= **false**)^ ((currentdate? – dom3 servicerecords i) > **90**) ^ (dom1 servicerecords i = dom1 creditcardinfo k) => creditcardinfo’ = creditcardinfo ⊕ { dom1 creditcardinfo k I---> **0** } )

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**INTERFACING WITH OTHER SYSTEMS**

**PART ORDERING SYSTEM**

The parts are ordered via Part Ordering System which takes the part name and the quantity needed and returns the delivery date of the part. Since the parts are ordered so only change in the system is in the inventory, which needs to be updated on the delivery date of the part. The other declarations used in the system remain unaffected by the Part Ordering System. Assuming that there is a helper function named UpdateInvenotry which takes the part id, quantity and date and updates the inventory on that day. The update inventory can function can be thought of as making use of the following predicate (where the declarations used will be from the schema below) :

∃i . (1 ≤ i ≤ |inventoryrecords| ) . (dom1 inventoryrecords i = partcode?) => inventoryrecords’ = inventoryrecords ⊕ { partcode? x dom2 inventoryrecords i I---> inventoryrecords i + quantity? }

\_\_\_\_\_\_\_\_\_\_OrderFromPartOrderingSystem\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

∆RoyalServiceStation

partcode? : PARTID

date? : DATE

deliverydate! : DATE

quatity? : QUANTITY

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

partcode? ∈ dom(products)

deliverydate! = PartOrderingSystem(partcode?, quantity?)

users’ = users

payment’ = payment

creditcardinfo’ = creditcardinfo

servicerecords’ = servicerecords

availablespace’ = availablespace

inventoryrecords’ = UpdateInventory( partcode?, quantity?, deliverydate! )

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**FUEL OREDERING SYSTEM**

The Royal Service Station interfaces with the Fuel Ordering System which asks for the type of fuel, date, station name, station code and gives back the delivery date. On the similar argument from the Part Ordering System here too I only need to update the inventory on the delivery date of the fuel. The same helper method is used here that was used in the Part Ordering System. The helper method might use this predicate for updating the inventory (the data type for the input used is described in the schema below):

∃i . (1 ≤ i ≤ |inventoryrecords| ) . (dom1 inventoryrecords i = gas?) => inventoryrecords’ = inventoryrecords ⊕ { gas? x dom2 inventoryrecords i ---> inventoryrecords i + gallon? }

\_\_\_\_\_\_\_\_\_\_OrderFromFuelOrderingSystem\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

∆RoyalServiceStation

gas? : PARTID

date? : DATE

deliverydate! : DATE

gallon? : QUANTITY

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

gas? ∈ dom (products)

deliverydate! = FuelOrderingSystem(gas?, gallon?, date?, **RoyalStation, 12345**)

users’ = users

payment’ = payment

creditcardinfo’ = creditcardinfo

servicerecords’ = servicerecords

availablespace’ = availablespace

inventoryrecords’ = UpdateInventory( gas?, gallons?, deliverydate! )

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CREDIT CARD SYSTEM**

The system is also capable of interfacing with the Credit Card System which takes card number, expiry date, name, and the amount of money and returns the transaction is accepted or not. IsParkingService function has been used to tell that the service was parking service or not. It takes service number as it’s input. If it is a parking service then the inventory level would not be affected. If it is either refuelling or the Vehicle Maintenance then the inventory level would be affected. Then in the latter case I looked for the corresponding inventory record and modified the amount of the quantity of the products by the amount consumed by the user. If the parking service is used then the available parking space will be reduced by one if there is available parking space in the parking lot. The user, park, product, creditcardinfo will remain unaffected. If the transaction is not approved then there will be no change in the schema except for the user where the last activity day is modified. The paid is set to true and the payby is set to credit if the transaction is approved. accept! represents whether the transaction is approved or not. The service! is used to check whether the service used is parking service.

\_\_\_\_\_\_\_\_\_\_AcceptTransaction\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

∆RoyalServiceStation

accept! : ACCEPTANCE

cardnumber? : ℕ

expirydate? : DATE

name? : STRING

amount? : PRICE

user? : USERID

product? : PRODUCTID

serviveno? : SERVICEID

paymenttype? : PAYMENTMODE

quantity? : QUANTITY

parkingtype? : PARKTYPE

date? : DATE

servicetype! : ACCEPTANCE

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

user? ∈ dom(users)

accept! = CreditCardSystem ( cardnumber?, expirydate?, name?, amount?)

users’ = users ⊕ { user? I---> date? }

creditcardinfo’ = creditcardinfo

(accept! = **false**) => (servicerecords’ = servicerecords) ^ (payment’ = payment) ^ (inventoryrecords’ = inventoryrecords) ^ (available’ = available)

servicetype! = isParkingService ( serviceno? )

(accept! = **true** ) ^ ( servicetype! = **true** ) => ( servicerecords’ = servicerecords U { user? x serviceno? x date? I---> amount? } ) ^ ( payment’ = payment U { serviceno? x **true** x **credit I--->** paymenttype? } ) ^ ( inventoryrecords’ = inventoryrecords ) ^ ( (availablespace’ = availablespace – 1) ^ (availablespace’ >= 0 ) )

(accept! = **true)** ^ (servicetype! = **false**) ^ ∃ i .( 1 ≤ i ≤ |inventoryrecords| . ( dom1 inventoryrecords i = product? ) => ( servicerecords’ = servicerecords U { user? x serviceno? x date? I---> amount? } ) ^ ( payment’ = payment U { serviceno? x **true** x **credit I--->** paymenttype? } ) ^ ( inventoryrecords’ = inventoryrecords ⊕ { product? x dom2 inventoryrecords i I---> (inventoryrecords i – quantity?) } ) ^ (availablespace’ = availablespace ) )

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_