A Petrol Station designed with OCL Language

Junjie He, Heying Cai
jhe654@aucklanduni.ac.nz, hcai665@aucklanduni.ac.nz

Introduction

A petrol company owns a number of petrol stations. Each petrol station consists of a number of petrol pumps. Each pump can be reset by a supervisor, after which it can pump out petrol from a common store, recording at each stage the volume and cost of petrol pumped so far. When pumping is finished, the total cost is recorded by the supervisor and subsequently paid by the customer. Each pump in a petrol station sells petrol at the same price, although this price may vary between stations. The common store can be re-stocked by the petrol company at any time. The company supplies petrol at the same cost to each station. Records are kept of the total volume of petrol supplied to each station, as well as the outstanding amount owed by each station. The amount owed to the company by a station is paid when requested by the company.

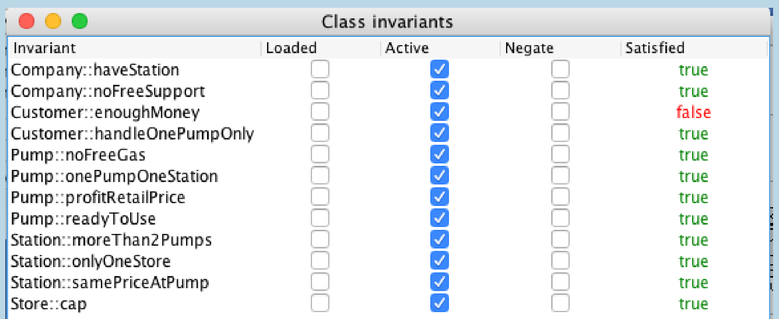
Detail Design Concepts

Some unique designs has been implemented into this petrol system. First, the pump will wait for superviosr to reset before it can online again. Second, supervisor will record the transaction every time a customer pumped gas from a pump. While he had reset every pump in the station, he will have a chance to report the Record he has to Station. And the Record at station side will be updated base on this. The Record at Station side is the one use to calculate how much station need pay to the Company. Since the company can collect bill anytime they want, station has been given operation resetRecord(), payOwe(amount:Real) and restock(volume:Real) to simulate the real scenario.

Verification with Invariant

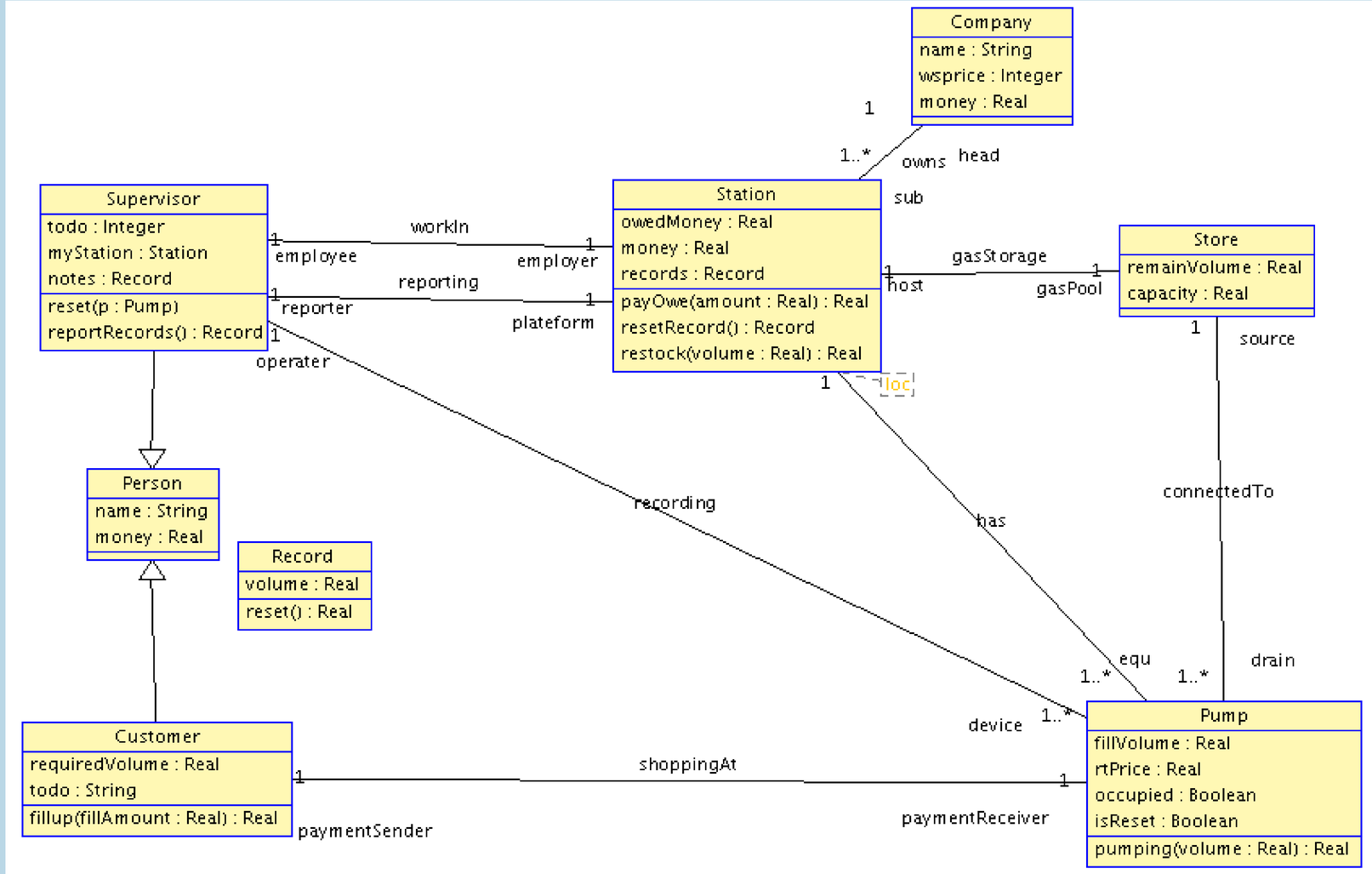
To verify the system invariant was add into UML specification to help with design. The invariant has been added into UML specification as verification. Notice the 'false' in red is intentionally left.

- *haveStation*The entire system shall at least have one station by a Company
- *noFreeSupport*The wholesale price a Company provide shall not be free
- *enoughMoney*A customer shall have enough money before engage
- *handleOnePumpOnly*The customer shall only interact with only pump a time
- *noFreeGas*The price on pump shall not be zero
- *onePumpOneStation*No common pumps cross station
- *profitRetailPrice*Check the retail price is profitable
- *readyToUse*Check if all pumps are ready to use
- *moreThan2Pumps*Each station shall have more than two pumps
- *onlyOneStore*Every station shall have and only have one store for storage
- *samePriceAtPump*This checks the sale price at pump station wides
- *cap*whether the storage is normal



Class Diagram

Class diagram demonstrate the relationship between classes and it can also provide the rolename for easy observation. Start from the head - company, they own stations and sale gas to stations with a wholesale price, they collect bill from station whenever they want. While that happens, Station::payOwe handles the situation. Pre/Post conditions are used to ensure Station have money to pay and Station holds a Record. This Record at Station get cleared after they pay off the bill.



Object Diagram

Partial view of object diagram is display here to demonstrate idea implementation of this petrol station. In this diagram we create instances for each class to demonstrate how the petrol station work. Taking slices of the cumulative 3D charts shows us how the degree distribution changes. The log-log charts below show the progression of these changes as the aggregation window gets larger.