Undergraduate Programme Academic Year 2014 – 2015 Coursework: Team Project

Module: CMP2515 Software Design UG2
School: Computing, Telecommunication and Networks
Module Co-ordinator: Professor Zhiming Liu
Setup Date: 20/02/2015
Submission Date: 22/04/2015

Team Leader: Callum Clarke

Team Members: Bobby Carver, Connor Bradley, Rhys Armstrong, Barham Khalil Ali, Mohamed Camara

Part 1: Requirements Understanding and Analysis

The Bvis Car Hire Company project that was designated concerns a car hire company that handles all of its data storage, including details of customers, employees and vehicles in the company fleet, using a completely paper-based system. In the modern day, where everything is computer-based in order to improve efficiency of work, reliability of data and overall professionalism of a business, this is a very archaic way of doing things. As such, our team needed to discuss the operation of this company and why using an object-oriented program design is applicable.

The most glaring point of all is of course the fact that object-oriented programming has become an absolutely massive force in the world of software; almost all programming projects today use object-oriented design over a procedural design, due at least part to the fact that this approach allows for much easier expansion, editing and maintenance of a program. This is especially true of projects such as this one where clearly defined classes are easily identifiable; hire, vehicle, and operator – these all would work extremely well with an object-oriented design, because it would be simple to find properties and responsibilities for all of these, including but not limited to the daily hire rate of a particular vehicle, the starting date and cost of a hire, and so on. These data could easily be applied to an object-oriented design.

批注 **[ZL1]:** A discussion is desirable about how the five attributes of complex systems apply to this project, such as being decomposable, evolving, intra- and inter-interactions of objects, etc

With this fact now identified, the basic operations that are required of the company system are listed here –

1. Register a new customer.
2. Record that a particular car has been hired.
3. Record that a particular car has been returned.
4. Calculate the cost based on the daily hire rate.
5. Display the appropriate details and print out a receipt.
6. Log a completed hire.
7. Record a service for a particular car, together with the date of the service, the type of the service and the name of the mechanic responsible.
8. Remove a customer.
9. Add a new car to the fleet.
10. Delete a car that is no longer in the hire fleet.
11. Add a mechanic who has joined the company.
12. Remove the details of a mechanic who has left the company.
13. Determine if a particular car is due for a particular service.
14. List the information about all hires for a specified car.
15. List the information about all services that a specified car has had.

Thinking about these required functions, use cases were identified by the team as follows.

Use Case	Register a New Customer
Actors	New customer, Operative
Purpose	Add a new customer's details to customer database system.
Overview	New customer wishing to hire a car goes to the Bvis outlet and enquires with a sales assistant. The assistant takes their name, phone number address and driving licence number and adds these details to the database.

Use Case	Hire Car
Actors	Operative, Current Customer
Purpose	Log the fact that a car has been hired out
Overview	Existing customer chooses a car and the hire start date, hire end date (estimate) and starting number of miles on the clock are recorded.

Use Case	Return Car
Actors	Operative, Current Customer
Purpose	Log the fact that a car has been returned
Overview	Existing customer brings back a hired car, and the actual return date and mileage are recorded.

Use Case	Calculate Hire Cost
Actors	Operative
Purpose	Calculate total cost of hire based on daily hire rate
Overview	The hiring cost of the car is calculated based on the daily hire rate of that car.

Use Case	Log Completed Hire
Actors	Operative
Purpose	Add hire details to company records
Overview	Record customer details, beginning and end date for hire, and payment costs and adds them to the company's records.

批注 [ZL2]: just Customer -- ``new'' or ``current" are about the state of a customer

批注 [ZL3]: ``data system" is a design decision not a requirement

带格式的: 突出显示

批注 [ZL4]: just Customer

带格式的: 突出显示

Use Case	Display/Print Details
Actors	Operative, Existing Customer
Purpose	Display details of hire period, including hire time, cost etc.
Overview	Details stored in the system are displayed on a screen for the operative to show to the customer, then a receipt is printed out containing the same details.

Use Case	Check mileage
Actors	Operative
Purpose	Check car mileage to see if it needs a service
Overview	The operative checks to see if there has been 6000 or 12,000 miles since the last service and then if so, send the car off for a minor or major service respectively.

Use Case	Record Service
Actors	Operative, Mechanic
Purpose	Log the details of a service
Overview	The type of service, the date of the service, the car it was performed on and the name of the mechanic who performed it are all logged in the company records.

Use Case	Remove a Customer
Actors	Operative, Existing Customer
Purpose	Remove a customer's details from the company records
Overview	If a customer has not used the car hire service for some period of time, then their details are removed from the company records.

Use Case	Add new car to fleet
Actors	Operative
Purpose	Add the details of a new car to the company records
Overview	When a new car is brought in to the company's fleet, the registration number, make, model, engine capacity, hire class, mileage date of last service and who performed it and date of registration are all logged and added to the company records.

Use Case	Remove car from fleet
Actors	Operative
Purpose	Delete the details of a car that is no longer in the company fleet from the company records
Overview	If a car is no longer in the company fleet then its details will be removed from the company records.

Use Case	Add a new Mechanic
Actors	Operative, New mechanic, Manager
Purpose	Log the details of a new mechanic to the company records
Overview	Operative checks if the new mechanic has a valid driving licence, then if so their name, address and home telephone number are added to the company records. The manager checks if the mechanic's details are unique, and if not, makes a note of this and gives an alias for that mechanic.

Use Case	Remove an existing mechanic
Actors	Operative
Purpose	Delete the details of a mechanic from the company records
Overview	If a mechanic leaves the company, then their details are removed from the company records.

Use Case	List hiring history/information
Actors	Operative
Purpose	Show the hiring history of a particular car
Overview	The operative brings up the entire history for all the hires a particular car has had.

Use Case	List service history/information
Actors	Operative
Purpose	Show the service history of a particular car
Overview	The operative brings up the entire history for all the services a particular car has had.

These use cases were then expanded on by the team (the workload was split out evenly in order to increase efficiency). The expanded use cases, as the name suggests, expand upon the top-level use cases by adding details of what should happen and in what order when a particular use case needs to be executed by the system – this involves detailing what the typical course of events is from the actor's point of view and how the system should respond, without going too deep into the inner workings. Alternate courses for if the events move away from the typical course are also taken into consideration. The expanded use cases are shown below –

Display/Print Details

Typical Course of Events	**System Response**
1. This use case begins after an **existing customer** has paid for the car rental.	
2. The **operative** uses the company records to display the rental information to the **existing customer**.	3. The system displays the **Customer details, beginning/end hire dates & payment costs** for the customer to see.
4. The **operative** then prints a receipt for the **existing customer**.	5. The system generates and prints a receipt.
6. The customer leaves with the details of the rental.	

Alternative Course:

Line 5: Printer is out of paper/broken, no receipt is printed. Replace paper/printer.

Add New Mechanic

Typical Course of Events	**System Response**
1. This use case begins when a **new mechanic** is hired	
2. **Operative** checks if the **new mechanic** has a valid driving license	
3. The **operative** then records the **new mechanic's** personal details.	4. **Name, address & telephone number** is added to the company records.
5. The **manager** checks if the **new mechanic's** details are unique and if not makes an alias on the system.	6. (If the **new mechanic's** details aren't unique) then the system will be updated.
7. **New mechanic** is hired.	logged

Alternative Course:

Line 2: If no valid driving license, the mechanic cannot be hired. Terminate hiring process.

批注 **[ZL5]:** The above part of the report show quite good understanding of the concept of use cases, good presentation of high level use cases. Try to avoid design details such as `database system'

批注 **[ZL6]:** why is this not part (or a sub-use case) of a larger use case such as `Return Car' (or Complete Hire)

批注 **[ZL7]:** remove

批注 **[ZL8]:** Better to state as ``a new mechanic is created, and the names, address &phone number" of the mechanic are set as provided in the details

批注 **[ZL9]:** should be alternative course of events

List Service History and Information Log

Typical Course of Events	**System Response**
1. This use case begins when the operative checks one of the car's service history	
2. Operative searches the system for a specific car and displays it's service history	3. **Service history** is displayed.
4. Operative can now end the search.	

Alternative Course:

Line 2: Details are entered incorrectly/the details entered do not exist in the company records.

User Action	System Response
1. The operative takes the customers details previously stored in the database upon registration, along with payment costs, start dates and end dates (all previously collected in past use cases).	2. The System outputs the data for the operative to use.
3. The operative then enters the details into the database and marks the hire complete, allowing for that car to be hired again.	4. The system checks that there are no duplicate entries which could cause errors.
	5. The system then marks the hire as complete within the database.

Restart the search process.

Log a Completed hire

Alternative Course:

Line 1) A different customer returns the car. In this situation the operative takes the original customer's details.

Line 2) The system is down. The operative will have to write down the customer's details on paper and inputs the data when the system is back online.

Line 4) If the details entered are duplicates (i.e. the car hire has already been marked as complete) the operative will take the car keys and not enter the completion into the database as it already exists.

User Action	System Response
1. The mechanic tells the operative details of the service. The details include date of service, car it was performed on (registration number), the mechanic who carried out the service's name and any notes that could be of interest (i.e. faulty parts, known damages etc...).	
2. The operative enters all of this data into the database which will be used to check when the next service will be due, hold the mechanic responsible if the service was not done correctly and to have a long list of service history when the car is no longer of use to the company and is sold on.	3. The system checks for the correct value type (such as integer, date etc…).
	4. The data is then stored in the systems database.

Record a Service

Alternate course:

2) The system is down. The operative will have to write down the service details on paper and input the data when the service is back online

3) If there are any errors with the input such as letters in cells that should be integers the system will display an error message and ask the operative to correct/check the data.

Register a New Customer

Typical Course of Events	**System Response**
1. Use case begins when a new, non-registered customer arrives at the Bvis outlet who wants to hire out a car, so they speak to an operative at their work station.	
2. The operative then takes the customer's details, and adds them to the company's database.	3. The new customer's phone number, address, name and driving licence number are added to the database. The system responds by presenting this information back to the operative to confirm its correctness.
4. The operative tells the system that the details are correct and the new customer's details are saved and registered to the company records.	5. The system saves the recorded details permanently into the records at this point.

Alternate Course:

Line 2: If details are entered incorrectly (such as in the incorrect format or data type) the system will ask the operative to correct/check the data.

Line 5: If the system is down at the time when the data is saved, then the operat5ive will have to write down the details on paper and enter them later.

Remove Car from Company Fleet

Typical Course of Events	**System Response**
1. An operative is told by either a mechanic or another operative that a car is being removed from the company fleet.	
2. The operative uses his workstation to remove the vehicle's details from the database, in this case using a system operation to do so.	3. The system deletes the vehicle's records and details, including registration number, mileage and from the fleet database.

Calculate Vehicle Hire Cost

Typical Course of Events	**System Response**
1) Daily car returned	2) Car returned recorded
3) The operative calculates the total cost	4) The system calculates the total cost of the hire based on the daily rate
5) The operative tells the total cost of the hired car to the customer and tells the customer can only pay by card	
6) Customer gives cash payment possibly greater than the total cost	
7) Operative records the cash received amount	8) System displays balance due back to the customer.
9) Operative keeps the cash and give back the balance owed to the customer	10) System records the date of the payment then prints out the receipt.

Alternative course:

Line 4: Customer doesn’t have the cash required, the sale cannot be completed. The customer has to get cash out from a nearby machine in order for the sale to be completed.

Add New Car to Company Fleet

Typical Course of Events	**System Response**
1) Starts when a new car is brought	
2) Operator checks the last usage and registration date of that new car	3) System the last performance of the usage and registration of that new car
4) Operator inputs the registration number, model, engine, hire class and mileage	5) The system records the each details of the new car and make it is ready to go
6) The operator assigned the mechanic responsible for the new car	7) The system keeps in records the last name and first name of the mechanic.

Alternate course:

Line 4: If data entered is incorrectly formatted, then the system will ask the operative to re-enter it.

Hire Car

Typical Course of Events	**System Response**
1) Use case begins when the Operative uses the computer to show the customer the cars and tells the operative for what starting date, ending date and he will check the current mileage on the car.	
2) The operative then takes the details, and adds them to the company's database.	3) The type of car, the starting data and ending data and current mile age, are added to the database. The system responds by presenting this information back to the operative to confirm it.
4) The operative reviews the details on the system for correction and then they are saved on the company records.	5) The system saves the recorded details permanently into the records at this point.

批注 **[ZL10]:** English is not clear

批注 **[ZL11]:** what details?

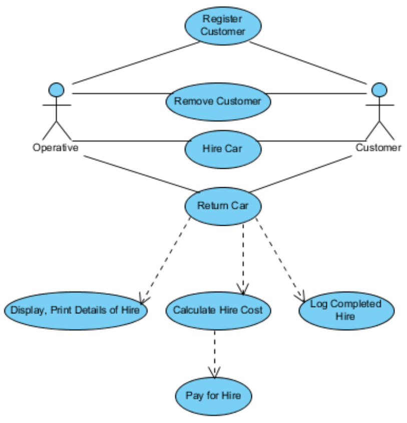
批注 **[ZL12]:** avoid design details

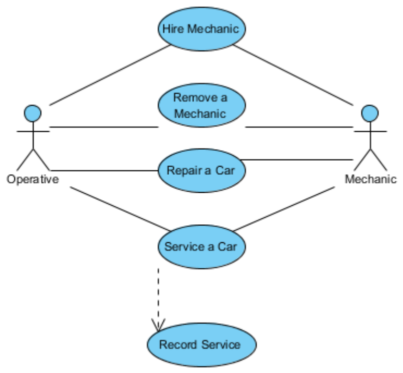
Remove a Customer

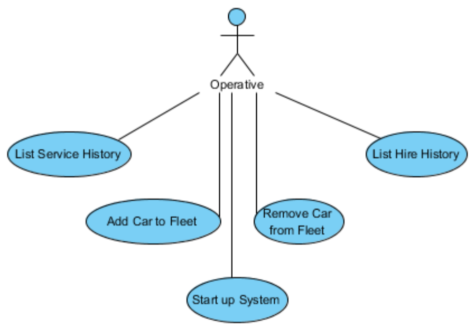
Typical Course of Events	**System Response**
1) Use case begins when the Costumer haven't user this Company for some period of time for instance a year. Then the company decides to remove the details of those customer from their system.	
2) The operative logs on to the computer reviews which customers haven't used there company to hire a car then clicks on the delete button to delete the customer's details.	3) The system response to the Operatives decision and deletes the customers details of the system
4) The operative reviews the systems once again to make sure the details of the customers is removed.	5) The system well no longer show any details under that detail.

It helps then to visualise these expanded use cases, including what actors are involved in each business transaction and what use cases are related to each other. Pictured below are the use case diagrams created by this process.

批注 **[ZL13]:** The expended use cases presented above show reasonable understanding of how interactions between customers. However, the understanding of the relation between use cases, and the execution of the interaction operations are mostly on input and output technology/mechanism, such as who to ``print" or " display" . Focus should be on changing of system states.



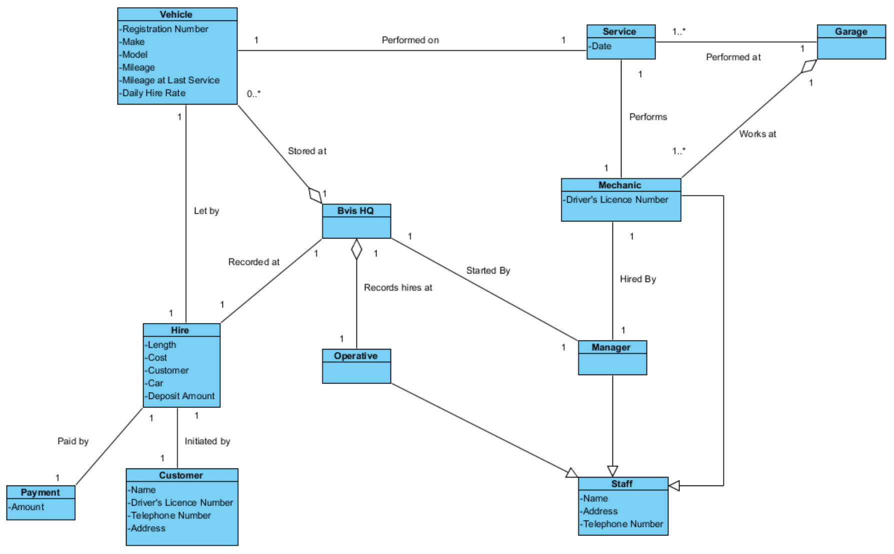




As can be seen, the use cases were grouped based on what actors are involved – sometimes there are more than one, but many of them only require the operative to take any action.

批注 [ZL14]: Use case diagrams are quite good, though could be improved

Having identified the key use cases and expanded upon them, the system workings can start to be identified at a slightly lower level. Pictured here is the conceptual class diagram for the system, which details how the use cases identified can be related to one another, including concepts of multiplicity and specialisation.



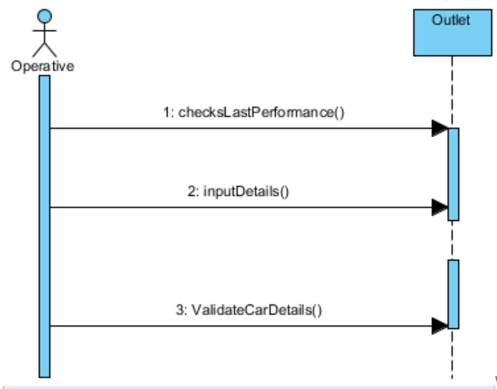
Some classes already have some attributes identified – this is to be expected. As stated earlier in this report, some elements of the system are noticeable immediately, such as the date of a service or the make and model of a vehicle. As progress is made in the design of the system, this diagram was likely to become outdated in terms of some of the properties, relations and even some of the identified classes shown. These changes can be represented

批注 [ZL15]: Classes, associations, and attributes are quite good. The understanding and use of generalisation-specialisation relations and aggregations are good. But several multiplicities are not right in lower level diagrams later in development, as those changes arise. Also it should be noted that some changes had to be made to the expanded use cases above after this diagram was finalised by the team, as some relations and other properties were not possible with regards to how they were described at that time.

Part 2: Functionality Analysis of System Operations

Following on from the above conceptual class diagram, the relationships between classes has been established, at least to some extent, as well as some properties of potential classes. To follow from this, the internal workings of the system can be discussed further. To start with, members of the team were assigned each some of the more important use cases, agreed on by the group, to create system sequence diagrams for. These diagrams represent a more in-depth view of how the actors involved in each system transaction would interact with the system and how it could respond.

Also included with each diagram are the contracts of the methods shown – these represent what state changes should be made as a result of the method being called or invoked, as well as what the overall responsibilities of that method or function are.



where is Outlet in your class diagram? or You need to explain so that this class will be added into the design class diagram

Add New Car to Fleet - Contracts:

Name: InputDetails ()

Responsibilities: it inputs all the details of the new car which included the registration number, model, hire class, last mileage, engine capacity and the make

Cross references: Use case: *Add new Car*

Exception: if the few car details are missing, indicate an error

Pre-condition: promptsDetails () is valid

Post-condition:

- If a new car is brought, a *add new car* is created
- When a new car is brought, *the add new car* details as (the registration number, model, hire class, last mileage and make) are associated to the *Outlet*

Name: ValidateCarDetails ()

Responsibilities: confirms that the new car details are right and proceed to make the car ready to be hired

Cross references: Use case: *Add new car*.

Exception: if the few car details are missing, indicate an error

Pre-condition: inputRegnumberModelHireClass () are inputted

Post conditions: The vehicle's details are validated by confirming with the operative that they are correct

The vehicle's records were committed to the company records

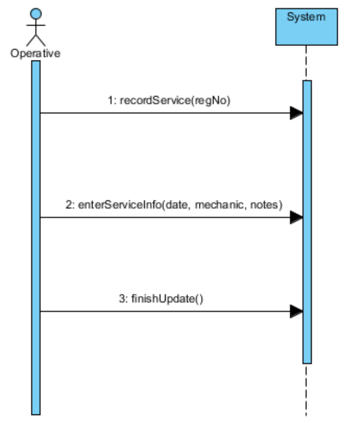
批注 [ZL16]:

批注 [ZL17]: what is this?

批注 [ZL18]: what does ``add" mean hear?

批注 [ZL19]: why not validate before create a new car?

批注 [ZL20]: This is not precise – new objects created, new links formed (or old links removed), or attributes of objects modified?



Record a Service - Contracts:

Name: recordService (regNo)

Responsibilities: This operation identifies a car from the database with its unique registration number.

Cross reference: Use case: Record a service.

Pre-conditions: A rental car has had a service which needs to be logged into the system.

Post-conditions: runs method: enterServiceInfo. New service created, assasiated with the unique vehicle.

批注 **[ZL21]:** ?

Name: enterServiceInfo (date, mechanic, notes)

Responsibilities: This operation updates the system with the details of the service that has been performed.

Cross reference: Use case: Record a service.

批注 **[ZL22]:** why is this not part of the recordService()

Pre-conditions: method: updateService has been ran and a registration has been entered

Post-conditions: Service information updated. Returns next service date.

Name: finishUpdate()

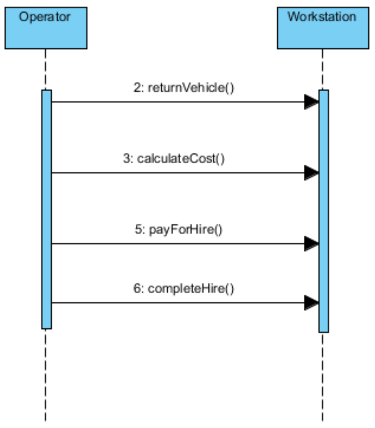
Responsibilities: exits the system.

Cross reference: Use case: Record a service.

Pre-conditions: N/A

Post-conditions: Exits the system.

批注 [ZL23]: This is not a ``condition “ on state



Return Vehicle – Contracts

Name: returnVehicle()

Responsibilities: After the hire is completed, log the fact that the vehicle is now available for hire again

Cross reference: Use case: Return Vehicle

Pre-conditions: The vehicle in question is currently on hire

Post-conditions: The vehicle’s status was set as ‘available’

Name: calculateCost()

Responsibilities: Calculate the final cost of the hire based on the duration of the hire, the daily hire rate of the vehicle and the change in mileage of the vehicle.

Cross reference: Use case: Return Vehicle

Pre-conditions: The vehicle in question has been hired for a period of time and then returned

Post-conditions: The total cost of the hire was calculated

The total cost of the hire was displayed to the operator

Name: payForHire()

Responsibilities: Tell the system and log in the company records that the hire has been successfully paid for.

Cross reference: Use case: Return Vehicle

Pre-conditions: The vehicle's total hire cost was calculated successfully

Post-conditions: After payment was made by the customer, the payment of the hire was logged as complete

Name: completeHire()

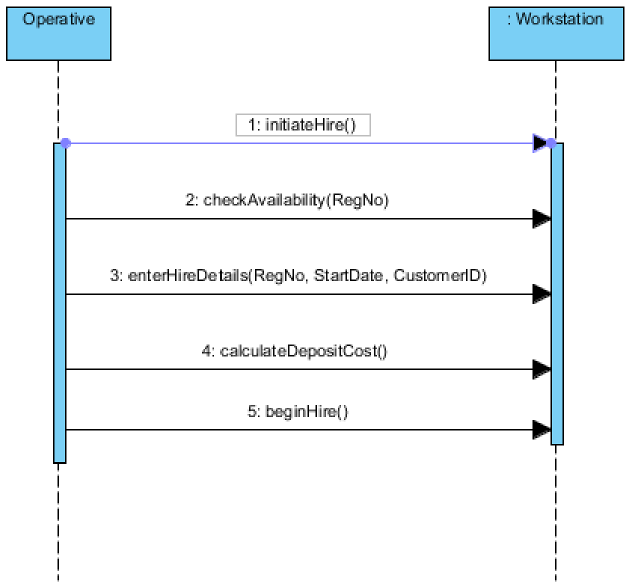
Responsibilities: Finalise the completion of the hire by logging it as complete and logging the vehicle as available

Cross reference: Use case: Return Vehicle

Pre-conditions: The vehicle has been hired and returned

Post-conditions: The hire status was set as complete

The vehicle status was set as available


Operative
: Workstation
1: initiateHire()
2: checkAvailability(RegNo)
3: enterHireDetails(RegNo, StartDate, CustomerID)
4: calculateDepositCost()
5: beginHire()

Hire Vehicle – Contracts

Name: initiateHire()

Responsibilities: Create a new hire

Cross References: Use case – Hire vehicle

Pre-conditions:

Post-conditions: A new hire object was created

Name: checkAvailability(RegNo)

Responsibilities: Check whether or not the registration number entered belongs to a vehicle that is available for hire

Cross References: Use case – Hire vehicle

Pre-conditions:

Post-conditions: The system identifies whether or not the vehicle is available, and if so continues with the hire process

Name: enterHireDetails(RegNo, StartDate, EndDate, CustomerName)

Responsibilities: Confirm the details of the hire including start and estimated end date

Cross References: Use case – Hire vehicle

Pre-conditions: A hire has already been created

Post-conditions: The properties of the hire were changed to match the customer's requirements

The hire was associated with the customer

Name: beginHire()

Responsibilities: Confirm the hire details with the operative and log the hire to the company records

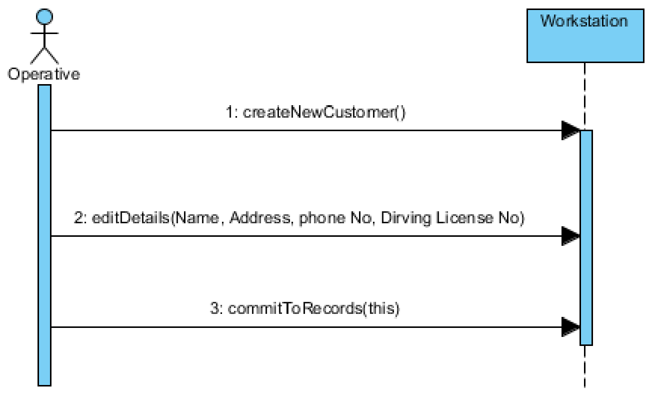
Cross References: Use case – Hire vehicle

Pre-conditions: A hire has been successfully created with no errors

Post-conditions: The vehicle status was changed to unavailable

The hire was logged as in progress in the company records

The hire was associated with the vehicle being hired



Add a New Customer – Contracts

Name: createNewCustomer

Responsibilities: Create a new customer for editing and committing to the company records

Cross References: Use case – Add a New Customer

Pre-conditions:

Post-conditions: A new customer object was created

Name: editDetails

Responsibilities: Add the customer's actual details to the object created

Cross References: Use case – Add a New Customer

Pre-conditions: There is a Customer object that can be edited

Post-conditions: The customer's details were altered to what the customer has told the operative

Name: commitToRecords(this)

Responsibilities: Finalise the new customer's details and add them to the company records

Cross References: Use case – Add a New Customer

Pre-conditions: The new customer's details are correct and the attributes of the customer object are set to these values

Post-conditions: The customer object was added to the company records

批注 **[ZL24]:** Part 2 shows the understanding of use case sequence diagrams, the modelling notation, and system operations, but the understanding of contracts of operations need to be improved

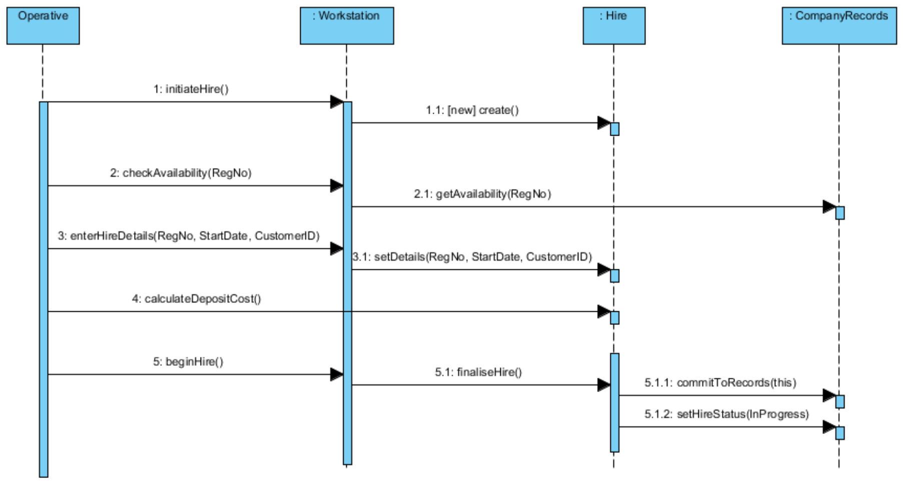
Part 3: Use Case Design

With these artefacts complete, the design for the system itself can begin. This is the point in development where design patterns are used; the expert, creator and controller patterns are the most commonly used in this project. For example, what class should be responsible for creating a new hire? Using the creator pattern, we can determine that the best choice for this would be the workstation class, using both the fact that the hire is closely used by the workstation and that the workstation records instances of them (indirectly, as it logs the hire to the company records).

Another instance of the use of design patterns in the project was the use of the expert pattern, the most common of all the design patterns. Though clearly this pattern is used all over the project because of its range of uses, but one of the clearest instances of its use would be when the team needed to determine what class should be responsible adding all the details of new vehicles, mechanics and hires to the company records? The solution, using the expert pattern, was the workstation, since this was the class that is most used by the operators at Bvis, and hence should have access to all the information necessary for the running of the company. All the classes – hire, vehicle and so forth – were then associated with the workstation directly or indirectly in order to follow this pattern.

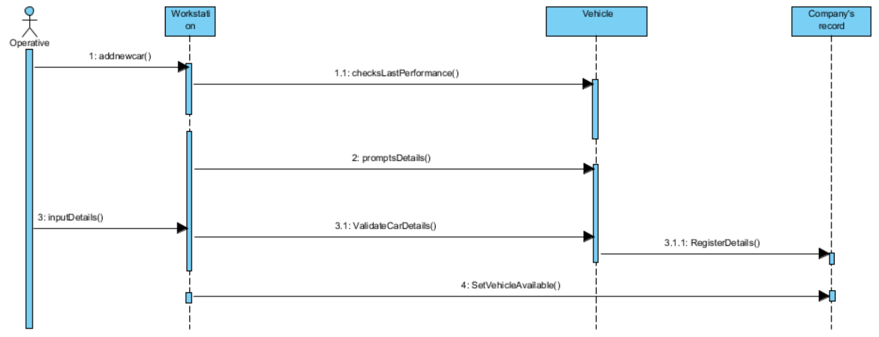
It should be noted that there is some disparity between the object sequence diagrams shown here and the final design class diagram later in the report with regards to things such as naming conventions and some relationship structures – this is due to the team deciding that some of these were incorrect or needed to some editing, so the names of some methods or attributes were changed for the final diagram.

The important use cases chosen from before were chosen for further expansion into object sequence diagrams, which show how the internal class-based components of the system should interact with each other during the operation of these use cases.


Operative
: Workstation
: Hire
: CompanyRecords
1: initiateHire()
1.1: [new] create()
2: checkAvailability(RegNo)
2.1: getAvailability(RegNo)
3: enterHireDetails(RegNo, StartDate, CustomerID)
3.1: setDetails(RegNo, StartDate, CustomerID)
4: calculateDepositCost()
5: beginHire()
5.1: finaliseHire()
5.1.1: commitToRecords(this)
5.1.2: setHireStatus(InProgress)

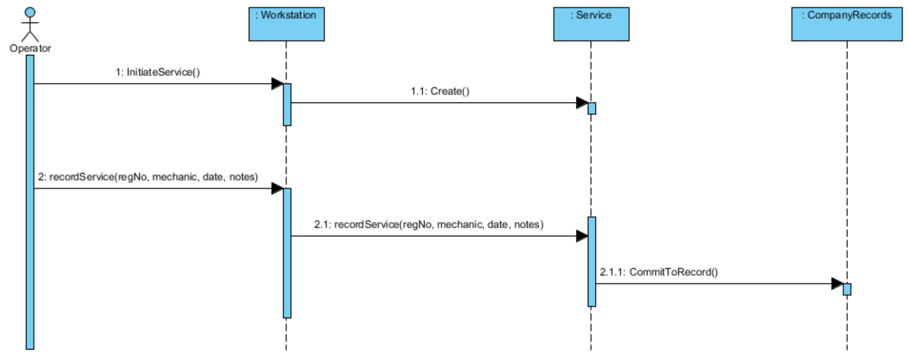
Object Sequence Diagram – Hire Vehicle

批注 [ZL25]: This is a quire good object sequence diagram



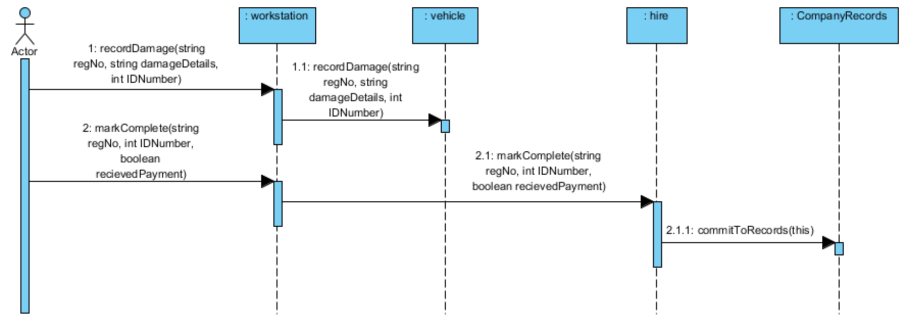
Object Sequence Diagram – Add New Car

批注 [ZL26]: 1.1. should``createVehicle", but in general, validate should be performed before create as otherwise the created instance should eb removed again if validation fails.



Object Sequence Diagram – Record a Service

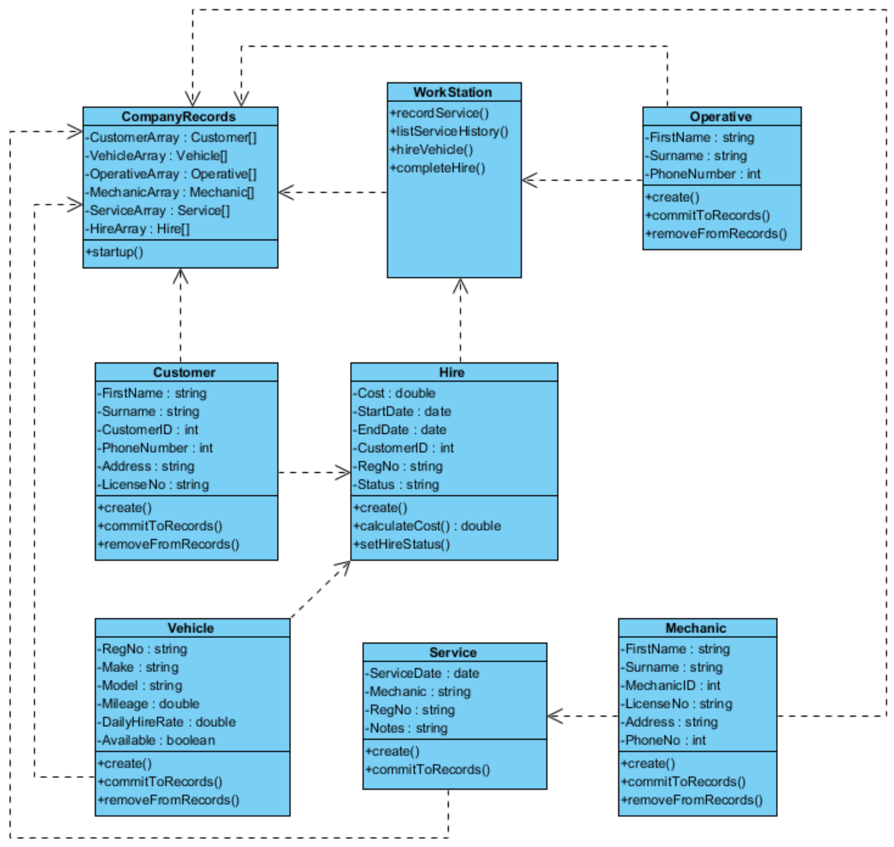
批注 [ZL27]: good



Actor
: workstation
: vehicle
: hire
: CompanyRecords
1: recordDamage(string regNo, string damageDetails, int IDNumber)
1.1: recordDamage(string regNo, string damageDetails, int IDNumber)
2: markComplete(string regNo, int IDNumber, boolean recievedPayment)
2.1: markComplete(string regNo, int IDNumber, boolean recievedPayment)
2.1.1: commitToRecords(this)

Object Sequence Diagram – Log Completed Hire

Having completed these diagrams, the final step in the design phase is the design class diagram. Essentially, this is an extended and updated version of the conceptual model from above in the report, having added further attributes and now introduced the methods (or modified versions of them) determined in the above diagrams and through other means. The naming conventions were also refined in order to conform more common programming conventions, such as camel-case method names, etc.



Final Design Class Diagram for Bvis Car Hire Company System

批注 **[ZL28]:** Quite good design from the requirements artefacts. However, the requirements could be improved with the feeadbacks of design. Consistency discussion, using of design patterns are not well discussed

Appendix

Glossary of terms

Multiplicity – the concept of how many of one class can interact with another, for example a many-to-one relationship or a one-to-one relationship.

Specialisation – When one class takes on the properties of another but adds its own unique ones, using the principals of inheritance from object-oriented programming.

Operative – The employee at the Bvis Company who performs a wide range of tasks including hiring out cars and adding a new mechanic to the company records.

Exception – Something that does not follow the predetermined course of actions and could cause an error to occur in the system.

Conceptual – The high-level form of something that can be refined later using various design techniques.

Team meeting schedule

Week, Date	**Attendees**	**Activities**	**Complete?**
1, 27/01/15	Callum, Rhys, Ali, Bobby, Connor	Establish group connections/communications. Discuss current company processes. Identify and create top-level use cases based on the user's needs.	Yes
2, 3/2/15	Callum, Ali, Bobby, Connor, Mohamed	Complete establishing basic use cases. Assign use cases to each group member to expand upon.	Yes
3, 10/02/15	Callum, Ali, Connor, Bobby, Rhys, Mohamed	Expanded use cases completed, discuss possible improvements to these and then unify them to a single document. Discuss use diagram and prepare to use team viewer and Visual paradigm tools to create it over the internet in real time.	Yes
4, 17/02/15	Ali, Mohamed, Bobby, Connor	Start on use case diagrams, complete alternate courses of action for all use cases.	Yes
5, 24/02/15	Callum, Bobby, Mohamed, Ali	Continue/complete use case diagrams, including relations between actors and use cases as well as relationships between use cases. Plan for conceptual class diagrams.	Yes
6, 03/03/2015	Callum, Ali, Rhys	Use case diagrams completed, continue work on conceptual class diagrams and models.	Yes
7, 10/3/2015	Callum, Ali, Rhys, Mohamed, Connor	Review conceptual model, identify which require system sequence diagrams and begin work on them.	Yes
8, 24/03/2015	Callum, Bobby	Continue work on system sequence diagrams and use case contracts	Yes
9, 13/04/2015	Callum, Bobby, Mohamed, Ali, Rhys, Connor	Finalise system sequence diagrams and contracts then begin work on object sequence diagrams	Yes

10, 20/04/2015	Callum, Bobby, Mohamed, Ali, Rhys, Connor	Finalise object sequence diagrams, then complete class design diagram and final report	Yes