Car Dealership Inventory Subsystem

Tyler Morrison
tjmorrison@cpp.edu

Emily Villalba
ervillaba@cpp.edu

Justin Robinson
jprobinson@cpp.edu

Ethan Bangar-Martinez
ethanb@cpp.edu

ABSTRACT

Inventory management is one of the most essential components in keeping a car dealership up and running. This process can be inefficient and prone to error if taken care of by fully human effort, so it is important to design a system that will satisfy the organizational needs of a car dealership business. We propose an inventory subsystem whose purpose is to access and keep track of inventory levels in a car dealership. The program aims to perform actions such as allowing employees to update the stock and manage when and by who cars are bought. Using information gathering techniques, user goal technique, domain modeling, and database design, our team developed a local database that provides functions such as filtered searches, empty stock notifications, and the ability to add or remove vehicles.

I. Introduction

Inventory tracking is a unique challenge that car dealership businesses face. The wide variety of makes, models, and hundreds of customizable features that are made available every year makes it seem as though there is an endless number of vehicles that must be documented and stored. The problem at hand is that recording how many cars are available to be sold, searching which cars are available at the dealership, and how many cars have been sold, and by who is a intensive process if done with manually with paperwork. It can also be inefficient and prone to error if taken care of by human effort, so our team proposes a subsystem that will satisfy the organizational needs of a car dealership business.

Our solution is to design a local database that will provide a way for the dealership to have an easier time accessing the inventory of the entire dealership as well as see when to order more stock. By researching, using our real world understanding of how dealerships operate, and gathering information from some of the resources found, we followed what we think a car dealership employee might need assistance from an inventory subsystem on an average workday. We decided to include functions such as filtered searches, ability to edit vehicle stock and add new models whenever necessary, record when cars are sold or removed from the lot, and notifications when stock has reached zero of a certain vehicle type.

The report is organized in the order the work was broken down. We began by discovering and understanding all aspects of the problem by investigating the work of dealerships, listing stakeholders of the system, and establishing a narrative. Then, the system requirements will be outlined in order to identify use cases and lay out all of the functions that our diagrams must take account of. We will go on to explain the logic behind our diagrams and how we designed the components of the solution to our problem. Finally, we will briefly describe our implementation process and highlight the difficulties we encountered while deploying our solution to the inventory dilemma.

II. Analysis & Design Techniques

A. Information Gathering

Ideally, the best way to establish an accurate understanding of the tasks that an inventory employee needs assistance with from a system would be through interviews, surveys, or on-site observation of a workday. As students, our options for information gathering is limited to online resources. Due to the variety of ways a car dealership can handle their inventory management, it was difficult to gather sources from companies who use processes that are realistically implementable for the level of experience our team gained from an Object-oriented Design course.

Online sources enriched our understanding of dealership inventory systems. The website ACV Auctions, who are described as a “proven team of seasoned entrepreneurs & automotive professionals”, published an article about the best practices for automotive retail inventory management [1].

The most essential information that we felt would guide us in the design of this system that dealerships should adopt time-minded approaches to avoid having cars sit on a lot for too long, keep inventory fresh by acknowledging aged inventory, and conducting regular inventory analysis.

B. Stakeholders

Stakeholders are people who have an interest in the successful implementation of the system. Logically, car salesmen are among the most important stakeholder because these employees interact directly with the system and their paychecks are affected when cars are marked as sold due to commission rates. They are internal because they are within the organization, and operational because they regularly interact with the system. Managers are also internal, as well as executive because they are within the organization and they care about the success of it. Managers grant access to the subsystem, use it directly, and care that operations of the dealership are smooth.

Corporate is an internal and executive stakeholder because company heads such as the CEO care that all vehicle are being accounted for accurately. Investors are executive and external stakeholders because they do not directly interact with the program, but they have financial interest in the system as they benefit when sales are conducted and vehicles are organized in a way that will achieve the most profit.

C. System Requirements

The system will provide a method for the dealership to have an easy way of accessing the inventory of the entire dealership as well as order more stock. It is a local database, so it is not connected to the internet or any cloud device. A username and password should be created by the manager for each salesman, granting them access to the database which is hosted on the lot. The system will have a database that stores the stock of the make, model, condition, and the VIN of each vehicle. As a customer purchases a vehicle, the database will be updated by the salesman or sales system, and an instance of a sold car will be created. The database will store the condition of the vehicle and its service record and warranty. The price of each vehicle will either be set or negotiable. If a vehicle make and model is non serviceable to the dealership, it will be marked as expired in the database. If a vehicle make and model is in the dealership for more than a certain amount of years, it will be marked as expired in the database. The user should be able to search through the database by filtering car types and models, as well as seeing the stock of each vehicle. When a sale is made, if the stock of a certain car type has reached zero, the car salesman will receive a notification. The salesman can either react to the notification by placing an order on a separate subsystem and marking it as on order in inventory or leave its status as not on order.

Using these general requirements, we applied the User Goal Technique to identify use cases. Use cases are activities that the system performs, usually in response to a request by a user. This technique is the most common in industry because it is simple and effective. First, we identify all of the potential categories of users of the system and then describe the tasks the computer can help them with, which we have done above. The next step is to refine the tasks into specific user goals, such as “Add car” or “Change order status”. It is important to use verb-noun phrasing when developing use cases to not get confused with the names of domain classes and to make the description of events as simple as possible.

D. Diagrams

The use cases and a brief description of their function follow:

- Search VIN – Used to search for a specific vehicle by Vehicle Identification Number
- Search Make & Model – Used to search for all cars of a specific type
- Add Car Type – Used to add generic car model
- Add Car – Used to add a specific car
- Remove Car – Used to remove car from database
- Create Salesman Account – Used when manager gives salesmen access by creating their account and handing it over
- List Vehicles – List all, available or sold
- List Available Vehicles – List all non-sold
- Update Order Status – Used when restock request is sent or stock is received
- Notify No Stock – System notifies user when a type of car stock is empty

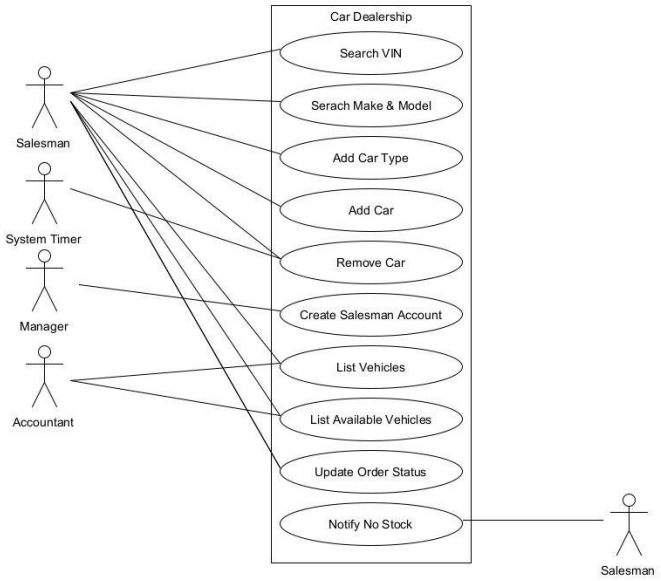
All the events are external except “Notify No Stock” and “Remove Car”. An external event is an event that occurs outside the system, usually initiated by an external agent or actor. “Remove Car” is external when an employee manually uses it to remove a car, but it is also a temporal event, an event that occurs as a result of reaching a point in time. After a defined period of time has elapsed, a car is now “expired” or “aged”, and it is time to remove it from the database and from the lot. There would be a notification that confirms its removal whether it is manual or automatic. “Notify No Stock” on the other hand is a state event. A state event is one that occurs when something happens inside the system that triggers some process. When the stock of a certain type of car reaches zero, the system will display a notification that indicates this information. It will also give the option to direct the user to either update the order status or leave it without modifying.

Use Cases

User/Actor	Event	Type
Salesman	Search VIN	External
Salesman	Search Make & Model	External
Salesman	Add Car Type	External
Salesman	Add Car	External
Salesman, System Timer	Remove Car	External, Temporal
Manager	Create Salesman Account	External
Salesman, Accountant	List Vehicles	External
Salesman, Accountant	List Available Vehicles	External
Salesman	Update Order Status	External
Salesman	Notify No Stock	State

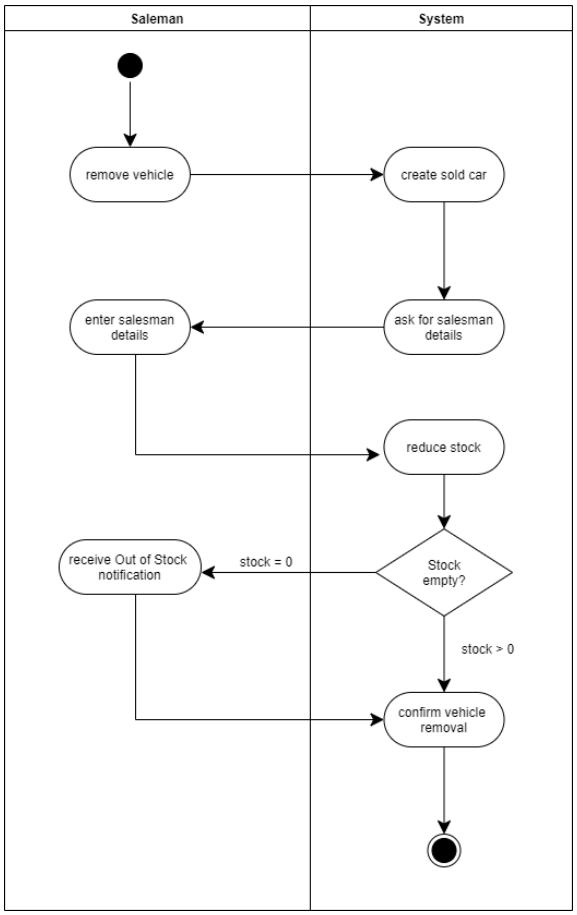
The following Use Case Diagram is a UML model used to graphically show use cases and their relationships to actors. The actors in this system include people who might request to view the data in the inventory database. Salesman are the primary users of the entire system, so they would be related to every single one except creating their own account. The manager is the one who creates the salesman account as a point in the hiring process, they hand over the account information as the key to the database. The salesman appears again on the right side. The actors on the left side of the automation boundary are acting on the system, while the actor on the right side is receiving information from the system like in the instance where the system notifies when stock is empty. The system timer is an actor that triggers the “Remove Car” case when a certain amount of time has passed, this amount of time can be modified by the user.

Use Case Diagram



Because the process of removing a vehicle from the database when it is sold is interesting, creating an Activity Diagram helped visualize the sequential flow of the activities. The salesmen initiate the process by indicating they want to remove a vehicle. The system creates an instance of a sold car which hold the time of the sale. Then the salesman is prompted to enter their details so that the sale will be associated under their account and they can get their commission later on. The system then reduces the stock of the vehicle type. It will analyze the stock levels. If the stock is equal to zero, the user will receive a notification that displays the information that suggests the vehicle is out of stock. The user will react to the notification and then the system will confirm the vehicle removal. If the stock was not empty, then it goes directly to the confirmation.

Activity Diagram for Removing Vehicle



The next step was domain modeling. We utilized the Brainstorming Technique, which involves creating a checklist of all of the usual types of things typically found within the topic at hand and creating classes accordingly. This is than hold tangible things you store information about, locations involved, or even roles played by people that we need to remember. Finally, we polished the list by including only the things the stakeholders truly care about.

The things we determined our system would care about storing the information of are:

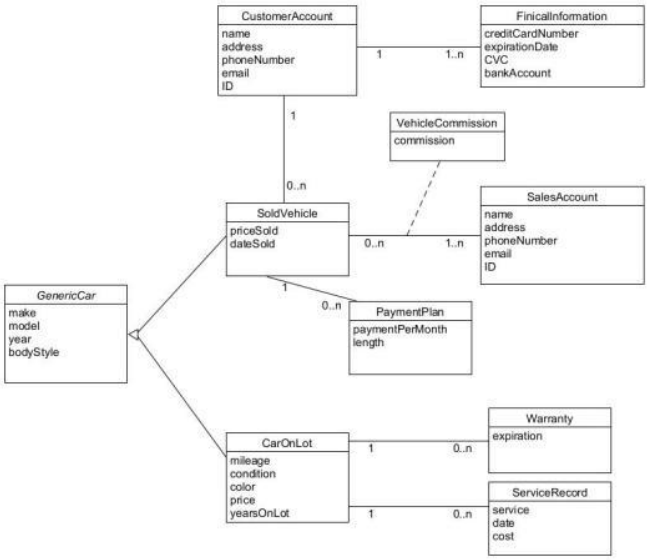
- GenericCar
    - CarOnLot
    - SoldVehicle
- PaymentPlan
- CustomerAccount
- SalesAccount
- VehicleCommision
- Warranty
- ServiceRecord
- FinancialInformation

GenericCar is interesting because after careful consideration, we decided to classify SoldVehicles and CarOnLots as types of GenericCar which is an abstract class, so all the attributes in GenericCar are passed down to SoldVehicle and CarOnLot.

Another interesting feature in the diagram is the association class between SoldVehicle and CarOnLot. This association was necessary because of the many to many relationship between the two classes. Many SalesAccounts can be connected to a SoldVehicle if there were multiple salesmen conducting a sale, and a SalesAccount can have many SoldVehicles connected to it. The association class allows this n to n relationship by associating them with the VehicleCommission, which holds the commission value.

Our domain class model diagram also includes some classes that we not to be implemented in the inventory subsystem because they would most likely be handed by another subsystem in the dealership. However, it was helpful to include on these classes on the diagram to show the multiplicities with classes that are involved in our system.

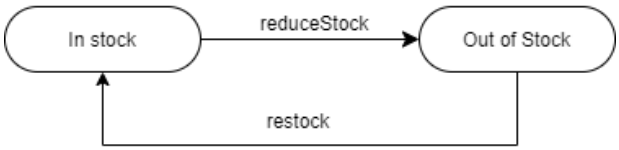
Domain Model Class Diagram



Each class has objects that may have status conditions or “states”. Object behavior consists of the multiple states and the transition between these states which are the conditions during an object’s existence when it satisfies some criterion, performs an action, or waits for an event. A state machine diagram shows the life of an object in states and transitions. We decided it would be helpful to design one of these machines to

explain the status changes of the GenericCar class objects.

A generic car type is either in stock or out of stock, being labeled as out of stock when quantity is reduced to zero after a certain amount of purchases.



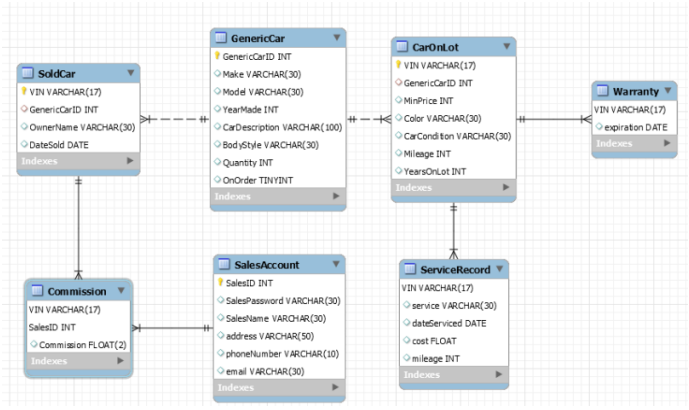
A generic car type is either not on order or on order. It is only on order when a salesman manually reacts to an Out Of Stock notification and sends an order to the manufacturer. The employee that sends out and order must mark it so that it is not reorder multiple times. When a new car of that type is added it is no longer on order.

III. Implementation Process & Team Findings

A. *Database Design*

Using some of team members' introductory knowledge to database systems, we utilized MySQL and Java Database Connectivity to create a program that allows access to data through a console-based user interface. Because our system is locally based and requires no connection to a cloud, a network diagram would be redundant as it would only include the server and the dealership's computers. We began by designing our database through a database schema which is a database component that contains descriptive information about the data stored in the physical data store.

This database schema we produced using MySQL includes only the classes in the Domain Model Class diagram that we actually implemented in the system. For example, FinancialInformation is not an essential component of inventory management so we did not consider it in the development of the database schema. Some interesting factor to point out is the use of foreign keys in the Car classes. GenericCar has its own GenericCarID that represents types of cars such as a Toyota Camry or Honda Civic. CarOnLot is a specific instance of a GenericCar, so that table has the GenericCarID as a foreign key and its own VIN as a primary key. The same goes for SoldCar. Another interesting factor is how we translated the association class. Because SoldCar and SalesAccount have a many to many relationship, the association table was required. This Commission table has the SoldCar's VIN and SalesAccount's SalesID as foreign keys, but they also serve as Commission's compound primary key.



One decision that we pointed out that could have been considered differently is the inclusion of a quantity attribute in the GenericCar. Essentially, a GenericCar could be a Toyota Camry 2020. An instance of CarOnLot would be a specific Toyota Camry 2020 with its own specific VIN that gets all the same information from the Toyota Camry 2020 GenericCar ID. We realize that the quantity attribute is not necessarily required because we could simply retrieve the count of how many CarOnLots have the GenericCarID as a foreign key. The database table would be more normalized if we did not include the quantity attribute. However, we decided to keep it for readability purposes and because some of the functions in the program depend on it. In a further iteration of our system, we would hope to work directly on making the database schema not just in First Normal Form (where all attributes values must be atomic, which we have done), but we would try to make it normalized in Second and Third Normal Form by eliminating these dependencies.

B. *Implementation*

Building and integrating the components was a rigorous task. Our team utilized JDBC's SQL connector to connect our MySQL Database to our java program. We then implemented MySQL commands to update add or remove pieces of data from our database. We used execute queries and a result set to run through our database's content and do what was necessary for each function. Whenever changes are made in the user's console, the data values were updated in the database once refreshed. For example, if the dealership were to have a White Toyota Camry 2020 Sedan, the car salesman would choose Add Car from the menu and

input the corresponding details as the program prompts for them. Our database would now store the details of this car under the CarOnLot table and increment the quantity of the Toyota Camry 2020 GenericCar. If a salesman were to remove a car when it is sold, this car would be removed from the CarOnLot table, decrement the value of quantity, and create an instance of a SoldCar and copy over all of the information connected to that VIN along with some new attributes such as DateSold. The user would input their name when prompted and their account would be linked to the sale as well.

Throughout the implementation process, we face some dilemmas. Due to time and knowledge constraints, we had to weigh some aspects of the final project. We decided to keep the user interface console based and focus on functionality over aesthetics. Because the only people who would use the program were employees and not the customers, have a beautiful interface was not a priority. However, if we had ample time, we would choose to create a Graphical User Interface to make the menus even friendlier. Despite this choice, we still made the console-based menus as readable and organized as possible.

We used our experience in Java programming language to create a menu-based program that made it easy to consolidate each functions' changes being made. The menu system we created made it simple to separate each function. When the user logs onto the system, they are presented with a menu that list the possible actions they could take. They would press 1 to add a car, 2 to remove a car, 9 to change order status of a car, etc. This was done with a switch case that alternated between different methods that would then be called and it would prompt the user for different information based on the requested task. Every add car, updated status, and search result was terminated with a confirmation from the system that ensured the task is done.

IV. Conclusion

A. *Difficulties*

Communication was a difficult task due to unforeseen circumstances that inhibited our ability to meet as a team, establish scheduled meeting times that worked for the entire team, and learning new technologies without being able to collaborate face to face. Once we were able to stay in touch, we struggled in the initial design of our system. Isolating the system's responsibilities was tricky because sometimes we could not decide if certain actions should be performed by the inventory subsystem or if it a task that should be reserved for sales or accounting or another part of the dealership. These issues were resolved by simplifying the actions and making sure only to implement things that involve quantities, reorders, and searching through existing vehicles.

We also struggled setting out realistic expectations for our skillset. Our main difficulties have revolved around trying to design a system that is realistically implementable. Initially we wanted to include an aesthetically pleasing and interactive graphical user interface to make our subsystem seem more complete. We also strived for more complex interactions with the manufacturers in which the subsystem would have the ability to contact a manufacturer directly when a certain car is out of stock instead of requiring the car salesman to act separately. Due to our limited experience with implementing actual databases, there were some hard times trying to figure out how to connect the modifications made with the program and actually having it update the data tables in MySQL. Luckily one of our team members is currently taking a course solely based around databases so their experience guided us with their comfort in MySQL.

B. *What Did We Learn?*

We learned a lot about communication and collaboration in the world of system design. Making sure our diagrams and our implementation match was a complicated task, but we gained practice by ensuring that people working on diagrams and those working on the coding aspect stayed consistent with the changing system requirements. It was important establish solid system requirements that make sense with our narrative early on so that there were no discrepancies between our initial system description and the program.

We had to gain experience with MySQL's database management system and Java Database Connectivity. Using online resources, our team was able to create a program that successfully interacted with the database as we proposed a car dealership inventory management team might benefit from. Bouncing ideas off each other and brainstorming through collaboration is essential skill that we practice throughout the entirety of the project. When we enter the computer science field as real programmers, we will have to learn how to work with teams on a certain subsystem that needs to interact with a larger system. These are skills that we gained from successfully accomplishing the creation this inventory subsystem that we will carry into the rest of our careers.

REFERENCES

[1] “Automotive Retail Inventory Management Best Practices for Dealerships”, ACV Auctions, ACV Auctions Inc, 2019. [Online]. Available: https://www.acvauctions.com/story/inventory-management-best-practices-for-car-dealerships/

[2] Mo Zahabi, “3 Ways Smart Dealers Manage Inventory”, Auto Dealer Today, 2018. [Online]. Available: https://www.autodealertodaymagazine.com/311179/3-ways-smart-dealers-manage-inventory

[3] Steve Lander, “How to Implement Inventory Management Systems”, Chron, Hearst Newspapers, 2020. [Online]. Available: https://smallbusiness.chron.com/implement-inventory-management-systems-80459.html

[4] “How to display or print the contents of a database table as is?”, StackOverflow, Hearst Newspapers, 2013. [Online]. Available: https://stackoverflow.com/questions/15444982/how-to-display-or-print-the-contents-of-a-database-table-as-is

[5] “Learn JDBC – Java Database Connectivity”, TutorialsPoint, 2020. [Online]. Available: https://www.tutorialspoint.com/jdbc/index.htm