Cruise Control Software Development Document Version 0.11

CS 347 Software Development Process

Professor Reza Peyrovian

April 18, 2020

Team Banana Sundae

Jeffrey Lee, Christopher Moon, Siddhanth Patel, Ryan Qin

I pledge my honor that I have abided by the Stevens Honor System.

Table of Contents

Cruise Control Software Document Version 0.02 **3**
- Executive Summary 3
- Introduction 3

Project Requirements Section Version 0.03 **6**
- User Stories 6
- Enumerated Requirements 7
- Statement of Need and Feasibility 9
- Bounded Statement of Scope for the System or Product 9

Project Requirements Analysis Modeling Version 0.03 **11**
- Use Case Diagram 11
- Use Cases and UML Sequence Diagrams 11
- UML CRC Model Index Card 19
- UML Activity Diagram 20
- UML State Diagram 20

Cruise Control Software Architecture Version 0.02 **21**
- Architecture Style 21
- Software Components, Connectors, Constraints 23
- Control Management 23
- Data Architecture 23
- Architectural Designs 25
- Summary of Issues 29

Formal Review Report **30**

Cruise Control Software Document Version 0.02

Executive Summary

Our goal is to create a cruise control software that is capable of being used in a real-life setting, such as a highway, while also providing both safety and reliability. The purpose of this cruise control software is to alleviate the stress of driving on a long road trip by letting the driver take breaks while driving. The software will maintain a set speed by using sensors to determine the speed of the car. Our approach to creating a safe and reliable cruise control software is by using the Agile Development methods to organize and note every step of the software development process, emphasizing quality and efficiency. The sprints and user stories will help ensure that we develop high quality products in a timely manner that meets all the requirements of the customers.

Introduction

Driving is a stressful experience that needs to be addressed. In today's society, many people often have to go on long commutes to work or travel far distances for road trips. This puts a lot of stress on the driver since they have to maintain constant vigilance throughout the entire trip while also having to keep a foot on the accelerator. In order to alleviate the mental and physical burden of driving for a long time, a cruise control software will let the driver take their foot off the pedal and worry less about manually maintaining a certain speed on long trips. Reducing the weariness of drivers not only makes their driving experience more comfortable and

enjoyable, but it also improves the safety of everyone driving on the road. Therefore, cruise control software is something every vehicle should have.

Cruise control software is a mission critical system that is essentially a software that acts in real time with hard time constraints. This software is a system that needs to be capable of maintaining the speed of the vehicle either physically or electronically. Due to the system being mission critical, cruise control needs to complete all the necessary tasks within a certain time without failure. If the system did not meet the hard time constraints, the performance of the system would be adversely affected, thus making it unreliable. This would put the safety of the driver and the passengers, the vehicle itself, and any surrounding vehicles at risk.

For the cruise control software, the main customer needs will involve the following: reliability, efficiency, and safety. Customers will want a cruise control software that always works reliably, ensures the safety of the occupants, and has a system that can be initiated and ended with convenience through the use of a button. Additionally, the driver will need full control of the vehicle while the system is running in case the driver wants to adjust the vehicle. Lastly, the use of the software should not be confusing to the customer and should have a user-friendly interface on the dashboard such that a first-time user can use the software without difficulty.

Since the software aims to provide comfort, safety, and reliability to the driver, it will have multiple features that come together to serve its purpose. First, the cruise control is initiated by the driver who must set the speed manually and then press a button to lock the cruise control to the set speed. Next, the software takes over and maintains the vehicle at the desired speed by taking readings from various sensors in the car. At this point, the driver can relax their feet to

keep their car at a steady pace, or they can press on the pedal to accelerate the car. However, once the driver stops accelerating, the software will slow the car down until it reaches the previously set speed. Finally, the cruise control can be shut off manually by pressing a button or automatically by braking. Other small features include a memory feature which resumes the set speed after braking and coast feature that reduces the set speed without braking. In order to ensure the safety and reliability of the product, we plan to apply numerous tests to ensure there are no bugs and issues in the system.

As this is the first version of cruise control that we have developed, there will be many areas to improve upon. We hope to use this as a base to create an even better system in the future. Using our experience from developing the cruise control software, we hope to build on what we learned to create a more advanced system such as adaptive cruise control. In the next version of the cruise control software, we want it to be able to react to the surroundings and appropriately slow down or speed up. We also want the software to be able to keep the vehicle inside the lane by itself. With these additions, the burden on the driver would be reduced even more, and the system would be much more useful.

Project Requirements Section Version 0.03

User Stories

User Name: Ordinary driver

Description: As a driver, I need a cruise control that can ensure my own safety and the safety of others in the vehicle.

Confirmation:

Functional:

- Can I start and end cruise control whenever I need to?

- Can I set the speed of the cruise control to what I want?

- Can I change the speed when the cruise control is already turned on?

Non-Functional: Security:

- Is the system safe and reliable?

- Does the system have a fast response time for unexpected situations?

- Is a Hard Real Time Mission Critical System set in place?

User Name: Truck Driver

Description: As a Truck Driver, I need a cruise control system that can ensure that the truck and its cargo are safely transported to its intended destination.

Confirmation:

Functional:

- Can the cruise control stay on for long periods of time on the road?

- Does the cruise control account for larger loads and the momentum of the truck?

- Can I adjust the speed whenever necessary?

Non-Functional: Security:

- Can the system endure constant use and sustain long times on the road?

- Can the cruise control react quickly in unexpected situations?

User Name: Speed Sensors

Description: A speed sensor needs the cruise control software to ensure uninterrupted and constant communication so that it can maintain safe speeds.

Confirmation:

Functional:

- Can the cruise control remain in constant communication with the speed sensors?

- Can the software adjust the speed accordingly based on speed sensor data?

- Can the software ensure no disruptions in its communication?

Non-Functional: Security:

- Can the software constantly maintain contact with the speed sensors during long sessions?

- Can the cruise control keep up with varying levels of speed?

Enumerated Requirements

1. Functional Requirements:
    1.1. The Cruise Control System shall be able to be turned on first by pressing the button “Cruise” on the end of the control stalk behind the lower right side of the steering wheel.
    1.2. After it is turned on, the Cruise Control System shall first request values from the Speed Sensor, Brake Sensor, and Engine Sensor.
    1.3. The Cruise Control System shall display the green cruise control icon in the instrument display to show the system is ready.
    1.4. The Cruise Control System shall set the speed throttle to match the input from the speed sensor.
    1.5. The Cruise Control System shall maintain the desired speed while the Cruise Control System is on.

1.6. If the driver accelerates at any point while the Cruise Control is on, the Cruise Control System shall yield control of the car speed to the driver and then reduce the car back to the preset speed after the driver releases the accelerator.
1.7. The Cruise Control System shall be turned off by pressing the button “Cruise” on the end of the control stalk behind the lower right side of the steering wheel.
1.8. The Cruise Control System shall be turned off by applying the brakes at any point during use.
1.9. After it is turned off, the green cruise control icon will disappear in the instrument display to show the system has been turned off.
1.10. The Cruise Control System shall be ready to be used again 2 seconds after the system has been turned off.

2. Non-Functional Requirements:
2.1. The Cruise Control System shall be available for use 2 seconds after the car starts up. (Design goal 0.5 seconds)
2.2. The Cruise Control System shall have a response time of 500 milliseconds (0.5 seconds) between the time the user presses the button to set the speed for cruise control and the car actually reaching the speed wanted by the driver.
2.3. The Cruise Control System shall have a Hard Real Time Mission Critical System.
2.4. The Cruise Control System shall not turn on when the driver attempts to set the cruise control speed at below 25 mph.
2.5. The Cruise Control System shall not turn on when the driver attempts to set the cruise control speed at above 75 mph.
2.6. If the driver attempts to turn on the Cruise Control at a speed below 25 mph or above 75 mph, the Cruise Control System shall display an error message on the dashboard.
2.7. The Cruise Control System shall rely on the car battery as its main power supply.

3. Input Requirements:
3.1. The Cruise Control System shall get feedback and input from the speed sensor.
3.2. The Cruise Control System shall get feedback and input from the brake sensor.
3.3. The Cruise Control System shall get feedback and input from the engine sensors.
3.4. The Cruise Control System shall get signal from pressing the “Cruise” button.
3.5. The Cruise Control System shall get input from the clock.
3.6. The Cruise Control System shall get signal from the brake pedal being pressed.
3.7. The Cruise Control System shall get signal from the accelerator being pressed.

4. Reliability Requirements:
4.1. The Cruise Control System shall have highly reliable hardware.
4.2. The Cruise Control System shall have software that works 99.999% of the time.
4.3. The Cruise Control System shall maintain a System Error Log to track all errors.

4.4. The Cruise Control System shall notify the administrators of the system by sending reports of System Error Logs.
4.5. The Cruise Control shall be updated annually at the end of the year to ensure that it will work for at least 10 years.
4.6. The Cruise Control software shall be updated at respective car dealerships.

5. Security Requirements:
5.1. The system shall have a firewall to help protect against potential software attacks.
5.2. The system shall have an updated anti-malware software to help protect against potential software attacks.
5.3. The Cruise Control System shall log an error report to the system if an issue arises.
5.4. Only the developers and testers of the Cruise Control System shall be able to access and modify the software.
5.5. The Cruise Control System shall continue to operate on battery power from the alternator in case of an unexpected engine shutoff.
5.6. The Cruise Control shall maintain a consistent, uninterrupted connection with the speed and brake sensors while the system is turned on.
5.7. The Cruise Control System shall only be updated through a USB port that is only accessible by an administrator.
5.8. The Cruise Control System logs and error reports shall only be accessible by an administrator.

Statement of Need and Feasibility

- The need that we want to address when creating our cruise control system is to provide drivers with a reliable cruise control system that can be used, ideally in situations when our users are traveling for a long period of time or are experiencing slight fatigue while driving. For the Cruise Control System, we will need to implement a “Cruise” button that can turn on and off the cruise control system. Another need is to have the software turn on and respond immediately after a command is made on the system by the driver. Considering the team members involved and the methods that we plan on implementing for the software development process, such as Agile, this project is feasible.

Bounded Statement of Scope for the System or Product

- For the project of the Cruise Control System, our main objectives are providing a software that will ensure the safety of the driver during the use of this product, provide stakeholders with updates and notify them about any potential changes that may be made to the function or design of the system, and develop an interface that will be easy for any driver to use.

Depending on the progress of the conventional cruise control system, we will most likely not be able to advance to the development of an adaptive cruise control system. A constraint that we may face is time. The time we currently have remaining is approximately three months. With team members attending other classes and extracurricular activities, we may face limited time to complete some sections of the project before the deadline.

List of customers, users, and other stakeholders who participated in requirements elicitation:

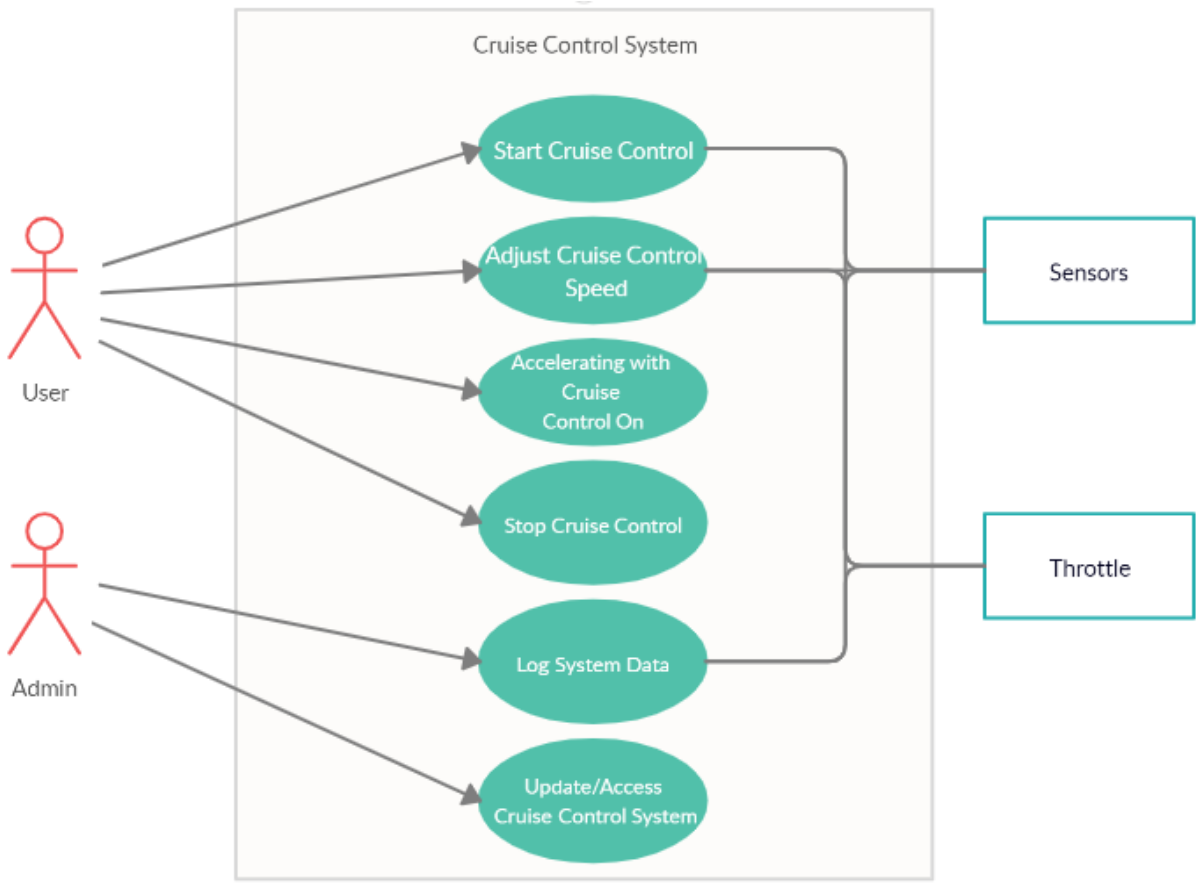
- Drivers in the Hoboken area
- Truck drivers
- Stakeholders (Jeffrey Lee, Christopher Moon, Siddhanth Patel, Ryan Qin)

Description of the system's technical environment:

- Hardware:
    - Basic mechanics of the car such as brakes, pedals, button, speed sensors, clock, and speedometer.
- Software:
    - Java
    - Possible IDE's: Visual Studio Code, Sublime, Notepad++

Project Requirements Analysis Modeling Version 0.03

Use Case Diagram



Use Cases and UML Sequence Diagrams

Use Case 1:

Name: Start Cruise Control

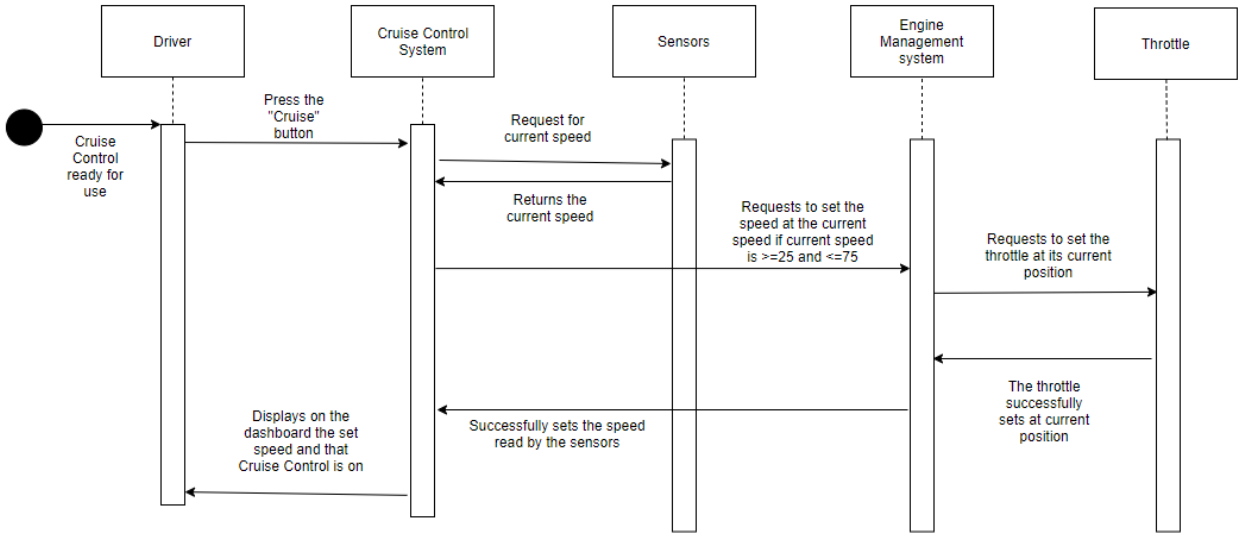
Brief Description: The user is driving and presses the “Cruise” button to turn on the Cruise Control System.

Actors: User

Basic Flow:

1. User is driving at least 25 mph and below 75 mph.
2. User presses the “Cruise” button.
3. Cruise Control System turns on.
4. Cruise Control System requests values from sensors.
5. Sensors provide values for approval to set cruise control at current speed.
6. Cruise Control System requests Engine Management System to set the speed at the current speed.

7. EMS Speed Throttle is set at its current position.
8. Cruise Control System provides visual feedback to the user that it is turned on and working.
9. Sensors provide the changing environmental information to the Cruise Control System (such as speed, request for increase/decrease speed, and brake).
10. Cruise Control System detects the changes from the sensors and requests to adjust the speed accordingly.
11. Speed Throttle position is continuously set to new values to ensure that the speed remains constant.
12. Speed is continuously reported to the Cruise Control System.



Use Case 2:

Name: Adjust Cruise Control Speed

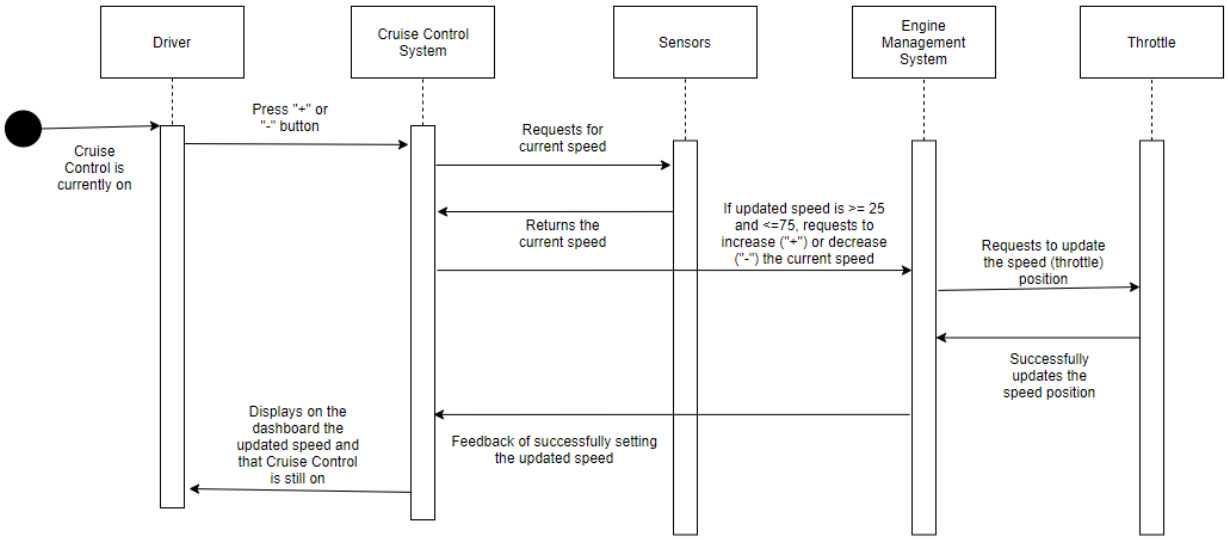
Brief Description: The Cruise Control System is on and working and the user adjusts the desired speed, keeping it between 25 to 75 mph.

Actors: User

Basic Flow:

1. Cruise Control System is on and working.
2. User presses the “+” button to increase the desired speed or the “-” button to decrease the desired speed.
3. Cruise Control System requests values from the sensors.
4. Sensors provide values to determine what the current speed is.
5. Cruise Control System requests the EMS to adjust the speed throttle position to match the new desired speed.
6. EMS adjusts the speed throttle position accordingly.

7. Sensors provide the changing environmental information to the Cruise Control System (such as speed, request for increase/decrease speed, and brake).
8. Cruise Control System detects the changes from the sensors and requests to adjust the speed accordingly.
9. Speed Throttle position is continuously set to new values to ensure that the speed remains constant.
10. Speed is continuously reported to the Cruise Control System.



Use Case 3:

Name: Accelerating with Cruise Control On

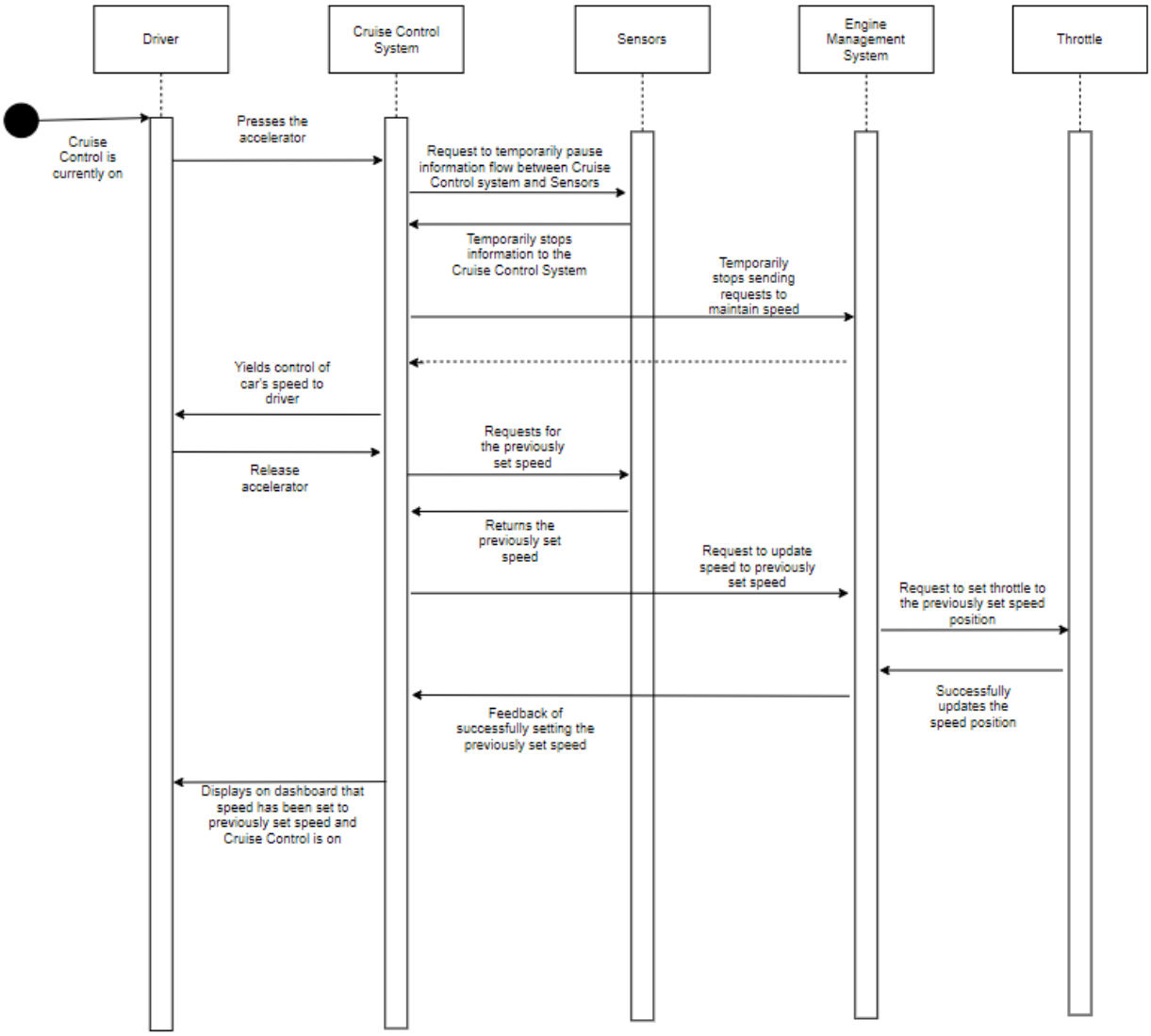
Brief Description: The Cruise Control System is on and working and the user accelerates.

Actors: User

Basic Flow:

1. Cruise Control System is on and working.
2. User presses the accelerator.
3. Cruise Control System yields control of the car's speed to the driver.
4. Cruise Control System stops requesting information from the sensors.
5. Cruise Control System stops sending requests to the EMS to maintain the speed.
6. User releases the accelerator.
7. Cruise Control System requests values from the sensors.
8. Sensors provide values to determine what the current speed is.
9. Cruise Control System requests the EMS to maintain the previous speed.
10. EMS decelerates the car to the previous speed throttle position.
11. Sensors provide the changing environmental information to the Cruise Control System (such as speed, request for increase/decrease speed, and brake).

12. Cruise Control System detects the changes from the sensors and requests to adjust the speed accordingly.
13. Speed Throttle position is continuously set to new values to ensure that the speed remains constant.
14. Speed is continuously reported to the Cruise Control System.



Use Case 4:

Name: Stop Cruise Control

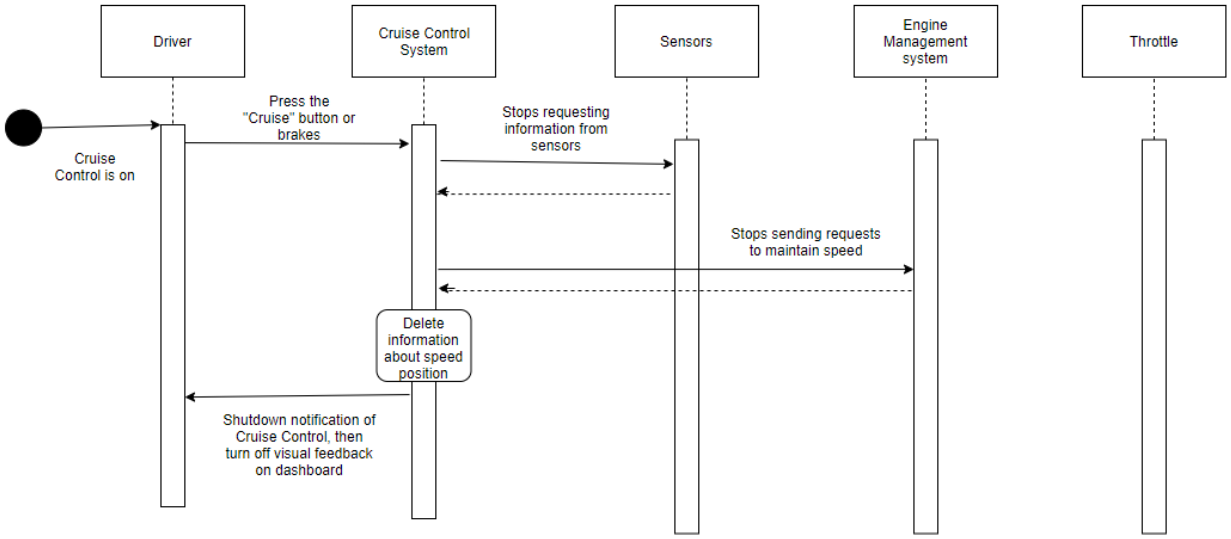
Brief Description: The Cruise Control System is on and working and the user turns it off.

Actors: User

Basic Flow:

1. Cruise Control System is on and working.
2. User presses the “Cruise” button to turn the System off or brakes to turn the System off.

3. Cruise Control System stops requesting information from the sensors.
4. Cruise Control System stops sending requests to the EMS to maintain the speed.
5. Cruise Control System deletes the information about the speed throttle position.
6. Cruise Control System turns off the visual feedback.
7. Cruise Control System turns off.



Use Case 5:

Name: Log System Data

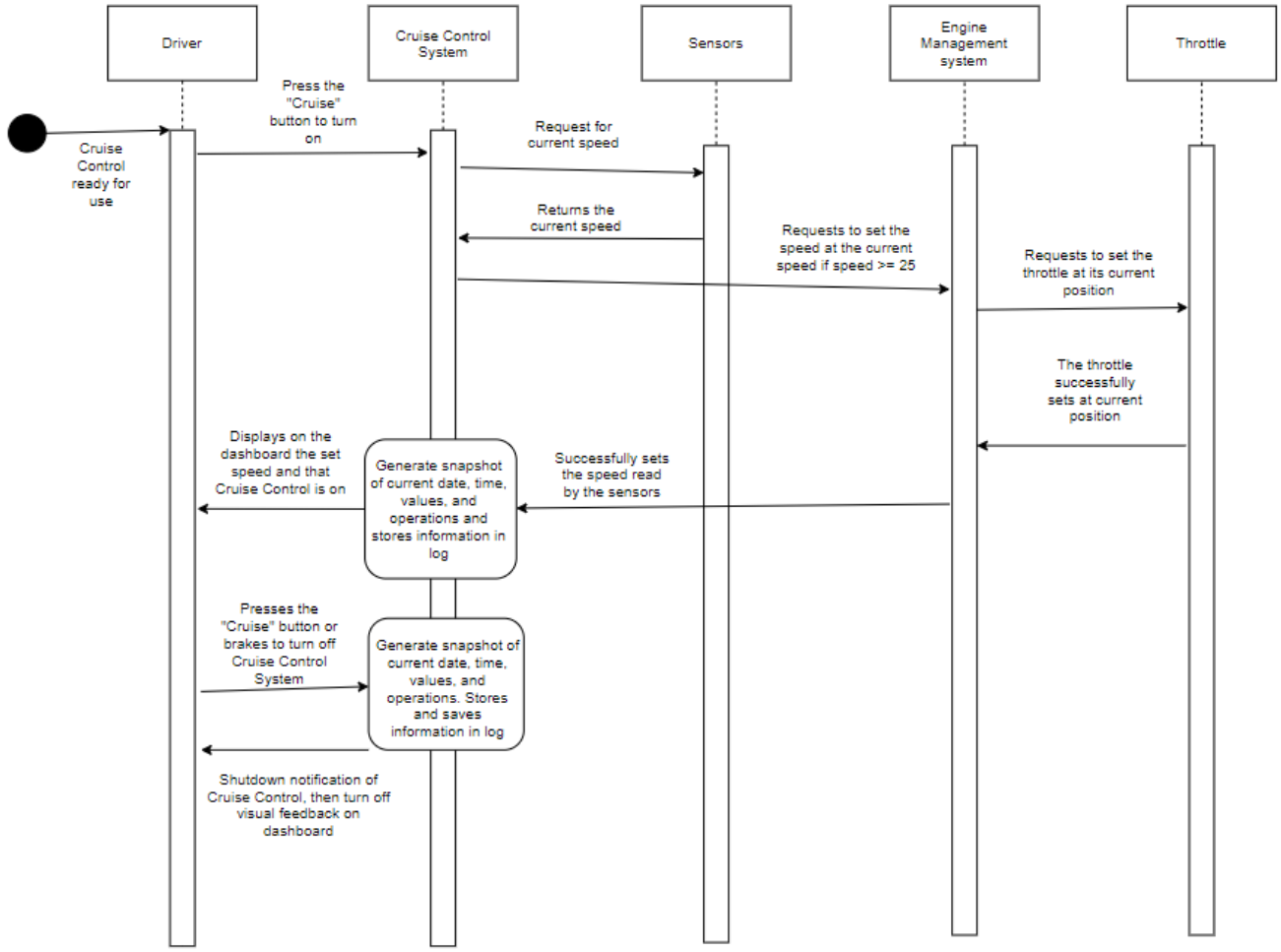
Brief Description: The Cruise Control System keeps track of the system information during usage and outputs a log of the information.

Actors: Cruise Control System, User

Basic Flow:

1. User presses the “Cruise” button to turn on the Cruise Control System.
2. Cruise Control System turns on and requests values from the sensors.
3. Sensors provide values for approval to set cruise control at the desired speed.
4. Cruise Control System requests Engine Management System to set the speed at the desired speed.
5. EMS Speed Throttle is set to the desired position.
6. Cruise Control System provides visual feedback to the user that it is turned on and working.
7. Cruise Control System generates a snapshot of the current date, time, values, and operations and stores the snapshot in the log.
8. Cruise Control System changes states (check State Diagram for the various states).

9. Cruise Control System generates a snapshot of the new date, time, values, and operations and stores the snapshot in the log.
10. Repeat steps 8 and 9 whenever the Cruise Control System changes state.
11. User presses the “Cruise” button or brakes to turn off the Cruise Control System.
12. Cruise Control System generates a snapshot of the current date, time, values, and operations and stores the snapshot in the log.
13. Cruise Control System saves the log.
14. Cruise Control System turns off.



<u>Use Case 6:</u>

Name: Update/Access Cruise Control System

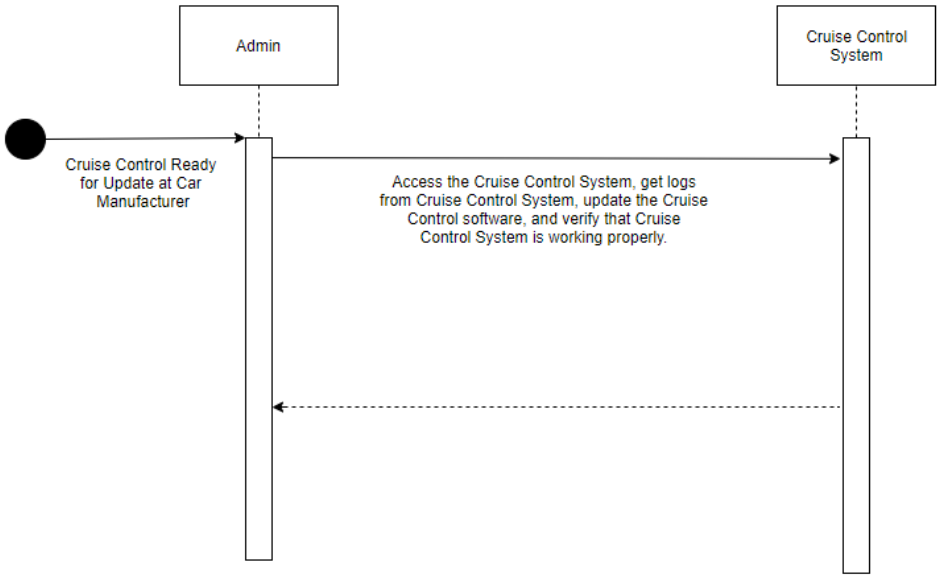
Brief Description: The admin accesses or updates the Cruise Control System.

Actors: Admin, User

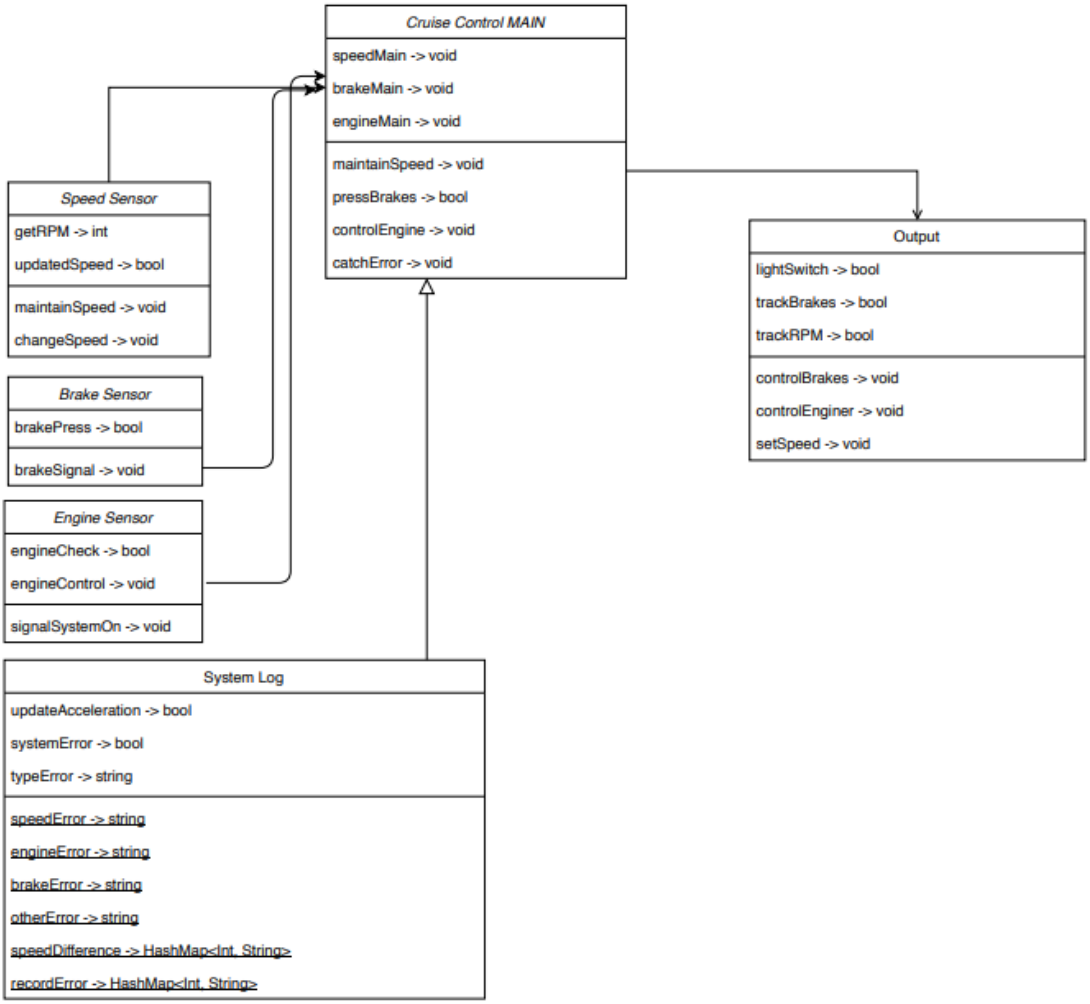
Basic Flow:

1. User brings the car to the car manufacturer.
2. Admin accesses the Cruise Control System in the user’s car.

3. Admin gets the logs from the Cruise Control System.
4. Admin updates the Cruise Control software.
5. Admin verifies that the Cruise Control System is working properly.
6. Admin returns the car to the user.



UML Class-Based Modeling



Relationships Between Classes:

- Speed Sensor & Cruise Control MAIN: Speed sensor provides information about the speed of the car to the Cruise Control System.
- Brake Sensor & Cruise Control MAIN: Brake sensor informs the Cruise Control System about the conditions of the brake (if brake is pressed or if brake is signaled)
- Engine Sensor & Cruise Control MAIN: Engine sensor provides information about the engine of the car to the Cruise Control System in order for the Cruise Control System to request necessary actions related to the engine
- System Log & Cruise Control MAIN: System log gives the Cruise Control System important information about the state of the system (speed, engine, brakes, etc.), especially if errors were detected in the system of the car
- Cruise Control MAIN & Output: Cruise Control System requests the actions needed to be taken to the Output class. The Output class will then complete the requested actions, such as setting the speed of the car.

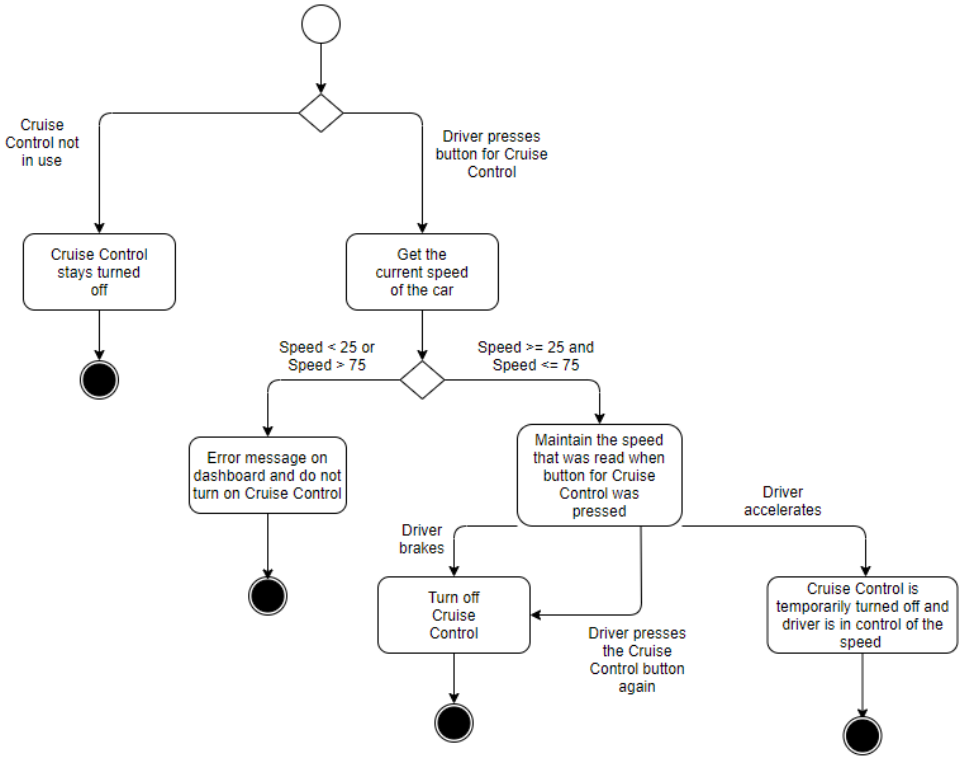
UML CRC Model Index Card

Class: Activity Turned On	
Represents the state of the cruise control when the system is on.	
Responsibility:	**Collaborator:**
Maintain speed at the desired level	Speed Sensor
Adjust cruising speed	
Turn off when brakes are pressed or when “Off” button is pressed	Brake Sensor, “Off” Button
Generate error logs	Error Logging System

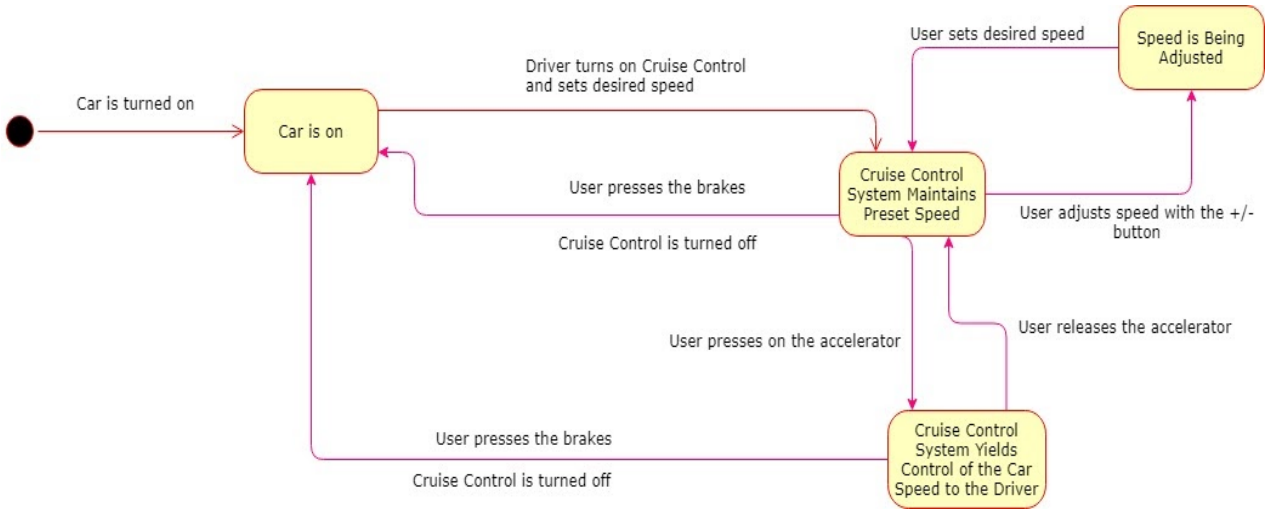
Class: Get Sensor Input	
Represents the state of sensors feeding back input	
Responsibility:	**Collaborator:**
Check the speed (rpms) of the vehicle	Speed Sensor
Check engine activity	Engine Sensor
Check if brakes are pressed	Brake Sensor

Class: Activity Turned Off	
Represents the state of the cruise control when the system is off.	
Responsibility:	**Collaborator:**
Release control of the speed	
Keep error logs safe and store in system	Error Logging System

UML Activity Diagram



UML State Diagram



Cruise Control Software Architecture Version 0.02

Architecture Style

Not Considered:

Data Centered Architecture

Data Flow Architecture

Layered Architecture

Considered:

Call Return Architecture

- Pros:
    - Ability to split main program into subprograms, such as having modules
    - Set hierarchy for the different programs if utilizing the main program and subroutine style of the Call Return Architecture
    - Can increase performance and improve efficiency by having various subprograms, which will split computations
    - Easy to modify
    - Easier to build on the program because of ability to add more subprograms
- Cons:
    - Difficult to have programs run in parallel
    - Have many different subprograms to manage
    - Usually fails to scale
    - Inadequate attention to data structures

Object-Oriented Architecture

- Pros:
    - Maps the application to real world objects for making it more understandable
    - Easy to maintain and improves the quality of the system due to program reuse
    - Provides reusability through polymorphism and abstraction
    - Has the ability to manage the errors during execution (Robustness)
    - Has the ability to extend new functionality and does not the system
    - Improves testability through encapsulation
    - Reduces the development time and cost
    - Modularity, extensibility, reusability, maintainability
    - Integrity: Data is manipulated only by the appropriate methods
    - Abstraction: Internals are hidden
- Cons:
    - Can be inefficient as more resources are being used due to how large the software can become
    - Difficult to determine all the necessary classes and objects required for a system

- Difficult to complete a solution within estimated time and budget because object-oriented architecture offers new kind of project management
- Does not lead to successful reuse on a large scale without an explicit reuse procedure
- Strong coupling between superclasses and subclasses. Swapping out superclasses can often break subclasses
- Not necessarily appropriate for smaller, less complex projects
- Not efficient enough for high performance computing (scientific computing, data science)
- Distributed applications require extensive middleware to provide access to remote objects
- In absence of additional structural principles unrestricted object-oriented programming can lead to highly complex applications

Model View Controller Architecture

- Pros:
  - There is a faster development process, because MVC supports rapid and parallel development.
  - Ability to provide multiple views for a model.
  - The architecture supports asynchronous technique, which helps developers to develop an application that loads very fast.
  - Any modifications in the application will not affect the entire model.
  - The model returns data without formatting, so it can be used and called for use for any interface.
- Cons:
  - There is high complexity to develop applications using this architecture.
  - Not suitable for small applications, which might have a negative effect in performance and design.

Most Relevant:

The most relevant software architecture for the Cruise Control System is the Object-Oriented Architecture. All developers in the group have unanimously decided that the advantages of using this structure for the Cruise Control outweigh those from the other software architectures. When developing the Cruise Control System, we will need to take into account the different classes, which for example, might be in charge of collecting input data, storing data, transferring data, and managing data. These classes can be represented best by objects which interact together in a structured manner to run the Cruise Control. An Object Oriented structure is easy to maintain and improves the quality of the system due to program reuse. Additionally, it has the ability to manage the errors during execution, which would be helpful for Error Logs in the Cruise Control Software. Other advantages include the ability to extend new functionality and improve testability which are vital for the Cruise Control, which will mandate periodic updates to its software. Therefore, the ideal software architecture for the Cruise Control System is the Object-Oriented Architecture due to its modularity, extensibility, reusability, and maintainability.

Software Components, Connectors, Constraints

Components:

The set of components include the various classes, such as the ones that collect user input data and store data, and a database where the data can be stored. They also include computational modules that are able to perform any methods required by the Cruise Control System, including maintaining speed and turning off when necessary, and objects that include data and associated operations.

Connectors:

In terms of connectors, the components are able to communicate with each other by sending data through message passing, which can also be seen as passing parameters to various methods in the system, and invoking methods from each of the classes.

Constraints:

Lastly, the constraints are defined as how the components can be integrated within the system. In this system, the classes and computational modules will serve to carry out system methods, the database would be used to store any relevant data, and the objects will be what is being passed around amongst the components in the system. Restrictions include the need to transfer data in real-time synchronously, the software being reliable and working 99.999% of the time, and secure data storage and transferring. Without these constraints in place, the software will not meet the requirements specified.

Control Management

Within the Object-Oriented Architecture, control is managed through the different classes, actors, and objects present. A distinct control hierarchy exists where each subclass contributes to the components of the superclass. This hierarchy is separated by tiers or levels. The components transfer control within the system by whatever call is made to a class. For instance, if a superclass calls for a method in a subclass, the control within the system will transfer from the superclass to the subclass until the subclass has successfully provided the superclass with the information needed. Because this hierarchy is separated by tiers, the control can be shared among components within the same tier. The control topology is a tree. There is one superclass on the top of the tree and the classes connecting below the superclass will be the subclasses in its own tier (e.g., Tier 2). Then, this pattern in the trees continues where each of the subclasses in Tier 2 will be the superclass for its connected subclasses in Tier 3. The components operate synchronized.

Data Architecture

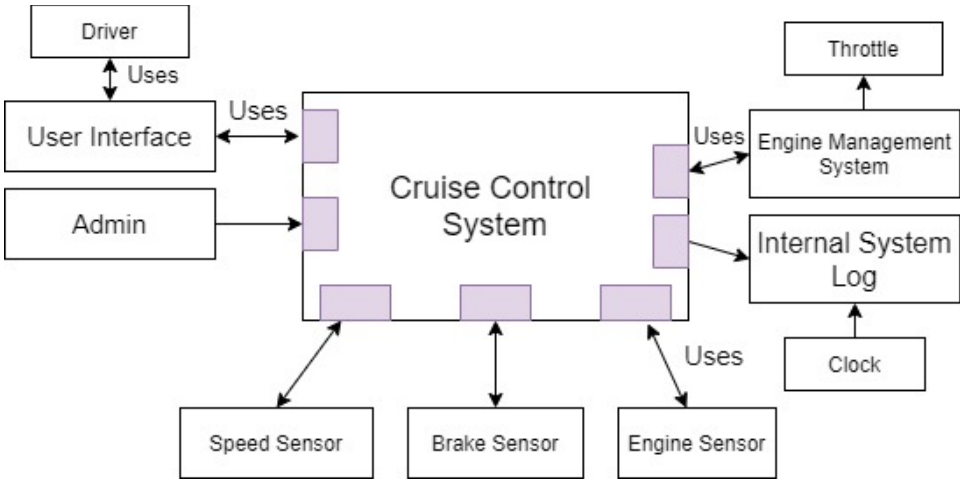
Data is communicated through the components by passing them as parameters to the program methods. The data is continually passed to the system. This is because the Cruise Control System needs to receive updates about the current speed so that it can maintain the desired speed. It constantly requests the EMS to adjust the throttle position accordingly. The

mode of data transfer is passing them in as parameters to the relevant methods. Data components provide information to the Cruise Control System which will then pass the relevant values to the relevant program methods. For example, the data about from the speed sensors would be given to the Cruise Control System and then the program would pass it to a method that asks the EMS to either slow down or speed up the car. Functional components interact with data components by taking the information from the data components and using them. Data components are active because the Cruise Control System has to actively request for values from the sensors and EMS whenever the Cruise Control System is updated.

There will be individual classes for the speed sensor, the brake sensor, and the engine sensor. The speed sensor will have the methods getRPM which returns an int, updatedSpeed which returns a bool, and maintainSpeed and changeSpeed which return voids. The brake sensors will have the methods brakePress which returns a bool and brakeSignal which returns a void. The engine sensor will have an engineCheck method which returns a bool, an engineControl method which returns void, and a signalSystemOn method which returns a void. The system log class will have an updateAcceleration method which returns a bool, a systemError method which returns a bool, and a typeError method which returns a string. There will also be methods that keep track of whenever the speed, brake, and engine sensors fail which will return strings. There will also be a speedDifference method that will return a hashmap of type <int, string> for storing the time of whenever the cruise control gets updated along with the updated speed. This will be used for logging purposes. The system log class will record all the error messages through a method called recordError that returns a hashmap of <int, string> for the time that an error occurs. The output class will have three methods that return bool: the lightSwitch method, the trackBrakes method, and trackRPM method. It will also have three methods that return void: the controlBrakes method, the controlEngineer method, and the setSpeed method. Lastly, the Cruise Control MAIN class will have methods relating to each sensor, the system log, and the output. The speedMain method is related to the speed sensor and returns void. The brakeMain method is related to the brake sensor and returns void. The engineMain method is related to the engine sensor and returns void. The maintainSpeed method returns void, the pressBrakes method returns bool, the controlEngine returns void, and the catchError method returns void.

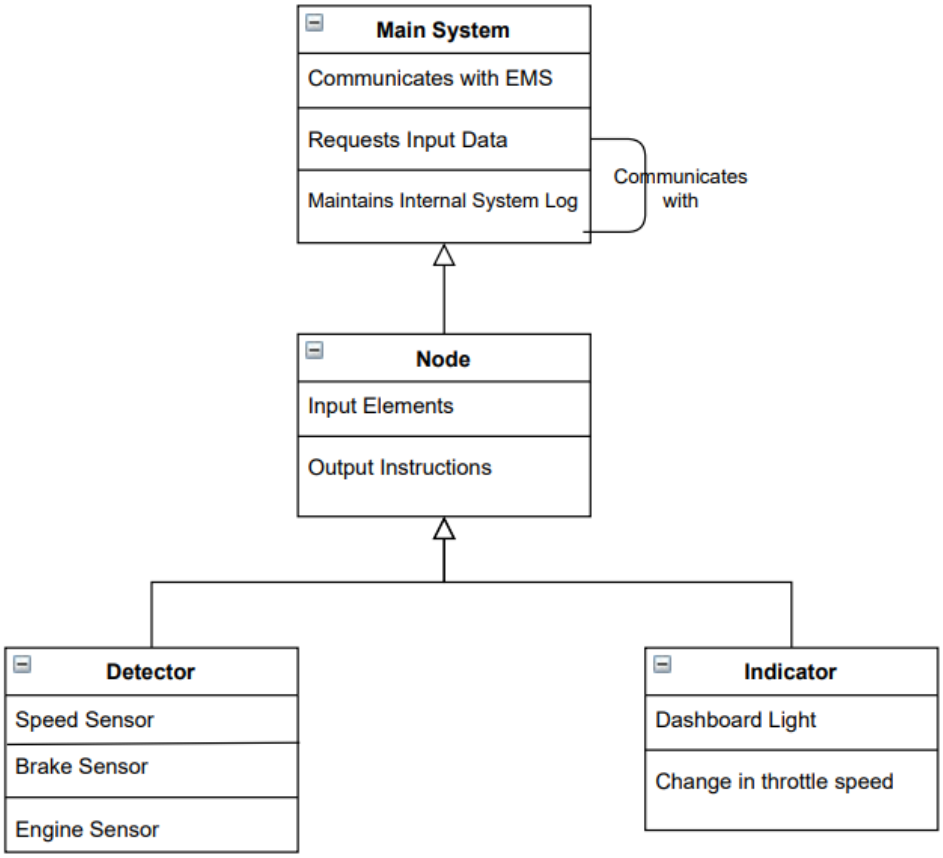
Architectural Designs

Architecture Context Diagram:



This architectural context diagram is a graphic representation of the Cruise Control System and the external components that interact with the system. The Cruise Control System will serve as the main object that connects internal and external components in order to maintain a smooth and seamless movement of input and output data. The single arrows indicate that one object is sending data to another object whereas the double arrows indicate that both objects are receiving and sending data to each other. The word “Uses” has been displayed on these double arrows to indicate the instances that both objects are constantly interacting with each other by sending and receiving data. For example, the driver inputs data into the user interface by pressing the “On/Off” button and receives data by the UI in the form of a dashboard light to indicate the status of the Cruise Control System.

Cruise Control System Archetype



This Cruise Control Archetype represents a core abstraction, that is critical to the design of an architecture for the Cruise Control System. The Cruise Control System architecture is composed of archetypes as defined:

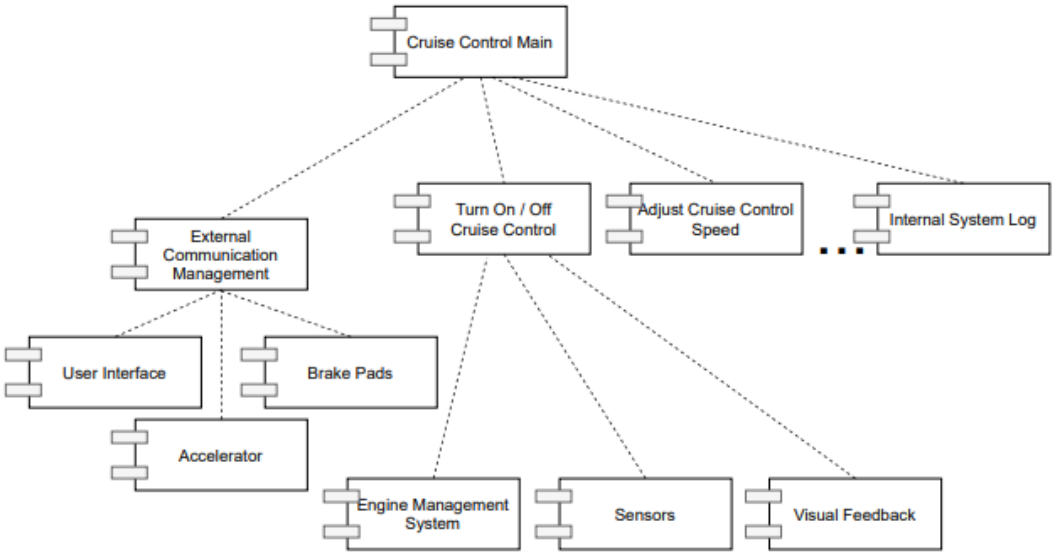
(i) Main System: Represents an abstraction that depicts the mechanism that allows the arming or disarming of a node. It uses the Node to communicate with the EMS, request input data, and maintain the internal system log.

(ii) Node: Represents a cohesive collection of input and output elements. It takes input elements from “Detector” (sensors) and maintains a list of instructions for the “Indicator” (UI and throttle speed).

(iii) Detector: Represents an abstraction that encompasses all sensing equipment that feeds information into the Cruise Control system. In this case, the speed, brake, and engine sensors send data to the main system.

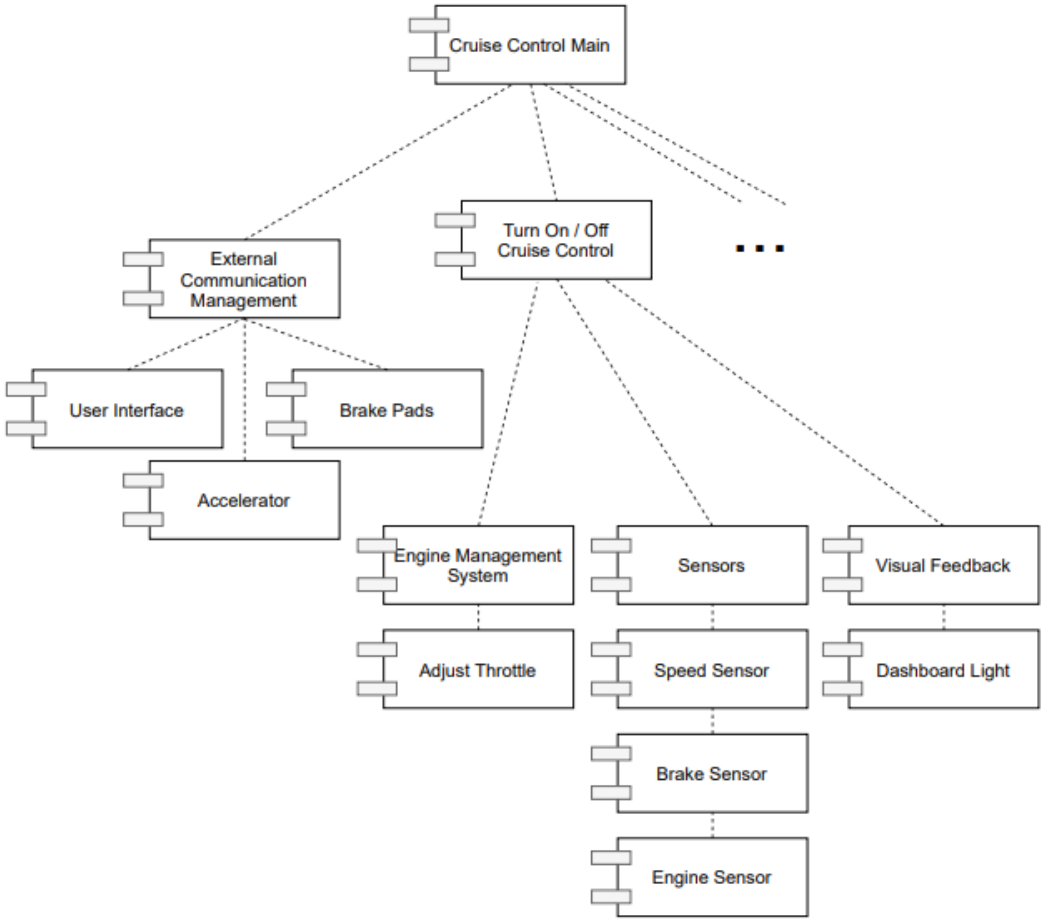
(iv) Indicator: Represents all mechanisms for indicating that the Cruise Control System is working. For example, the dashboard light and the change in throttle speed show signs that the system is in use.

Cruise Control Top-Level Component Architecture



The Cruise Control Top-Level Component Architecture displays the layout of all top-level components that make up the Cruise Control System. At the top, there is the Cruise Control Main object, which branches out into two general groups: the External Communication Management and the list of Methods/Functions. The External Communication Management is in charge of communicating with external components such as the UI, Accelerator, and Brake Pads. Meanwhile, the list of Methods/Functions is a group consisting of the various tasks that the Cruise Control System will accomplish, such as turn on/off, adjust speed, and maintain the internal system log. Finally, these functions branch off into other components which are required in each function. For example, the turn Cruise Control on/off component will branch off into the EMS, Sensors, and Visual Feedback components because it will require them to function. Overall, these branches connect all components together and represent a solid foundation for the top-level component architecture.

Cruise Control Refined Component Architecture



The Cruise Control Refined Component Architecture is an extension to the top-level component architecture. It displays the specific sub-components that are being used under each function/method in the Cruise Control System. For example, the EMS is further branches out to the throttle because the EMS plays a role in adjusting the throttle while the Cruise Control is turned on. Likewise, the sensors branch out into the specific sensors such as the speed, brake, and engine sensors. And finally, the visual feedback component consists of the dashboard light, which will indicate whether the Cruise Control System is on or off.

Summary of Issues

The main issue that we discovered is in regards to determining all the necessary classes of the system. Because not all of the classes are defined, it is not yet known if more classes will have to be implemented in order for the software to operate properly. This can create possible delays in the future when the software is being developed, which can impact the overall quality of the project. In addition, as we are following the Object-Oriented Architecture, if one of the superclasses is modified, then all the corresponding subclasses will need to be modified as well. This can cause a huge loss of time and resources. Thirdly, the objects of the system can be better defined in order to clear up any confusion. Lastly, the Data Architecture needs to be more thoroughly defined such that coding can begin.

Assignment of Issues:

1. Jeffrey: Not all the classes have been defined, so it is not known if more classes will have to be implemented in order for the software to operate properly. Work on generating most of the layout of the Cruise Control System.
2. Chris: Since if one of the superclasses is modified, then all the corresponding subclasses will need to be modified as well. Work with Jeffrey on generating the layout of the system in order to prevent any delay in the future.
3. Sid: Consider any other alternative architectures in order to ensure that the most optimal one was chosen. Also, the objects of the system are not clearly defined, so work on defining the objects. Lastly, schedule more team meetings.
4. Ryan: The Data Architecture is not well-defined. Work on defining any classes that are being used and clarifying any confusing aspects.

Resolutions:

- We made the Data Architecture more specific and refined. Therefore, after a long discussion, we have all agreed upon a complete list of classes for the Cruise Control System to operate properly.
- We have also clearly defined the order/priority of each object in the event to prevent errors with the transfer of data among the classes.
- During our discussion, we have drawn out a clear layout using the “Bridge” diagram as described in class. This diagram will serve as our framework to begin coding.
- We have considered efficiency in our coding process in order to account for the small timeframe that we have to produce the finalized product. We will aim to maximize quality of the software by reviewing all team members contributions to the code on a weekly basis.
- We will schedule more team meetings in the near future to resolve outstanding issues and any other complications that may arise in the development process.

Formal Review Report

Date/Time: 3 May 2020, Times: 3:00 PM - 4:12 PM EST

Participants: Jeffrey Lee, Christopher Moon, Siddhanth Patel, Ryan Qin

Preparation Effort:

The effort required to review the Cruise Control Software Document prior to the actual review meeting was on average **43 minutes per person**.

Assessment Effort:

The effort required during the actual review of the Cruise Control Software Document was **1 hour and 12 minutes per person**.

Rework Effort:

The effort dedicated to the correction of errors uncovered during the review was **56 minutes per person**.

Work Product Size:

The number of pages of the Cruise Control Software Document: **30**.

Findings/Issues List:

Minor Errors:

1. For UML class-based modeling, the relationship between classes is not clearly defined. The classes are related to one another in some fashion.
2. Requirement for sending requests to sensors before activation was missing.
3. Requirement for the power supply of the cruise control system was missing.
4. Not all the classes have been defined, so it is not known if more classes will have to be implemented in order for the software to operate properly.
5. Since if one of the superclasses is modified, then all the corresponding subclasses will need to be modified as well.
6. Consider any other alternative architectures in order to ensure that the most optimal one was chosen.
7. Inconsistent term usage for throttle and speed position in the use cases.
8. We did not specify what error statement would be displayed in the event of an error while using cruise control.
9. The Architecture Context Diagram needs to define what “Uses” and the double arrows signify.
10. The Cruise Control System Archetype needs to define what the different components are such as “Main System”, “Node”, “Detector”, and “Indicator”. Describe their relationship.

11. The Cruise Control Top-Level Component Architecture needs a summary to describe the relationship between the components.
12. The Cruise Control Refined Component Architecture also needs a summary to describe the relationship between the components.
13. The components, connectors, and constraints should be distinguished from each other.
14. Did not use the bridge diagram to serve as the framework for our object-oriented programming.

Major Errors:

1. System's technical environment was outdated and incorrect. We will not be using C++ anymore. Java is a more suitable software language for the Cruise Control Software.
2. The Data Architecture is not well-defined. It needs to be expanded and be more specific.

Conclusions/Resolutions:

We have all reviewed the Cruise Control Software Document extensively. The list of issues above has been thoroughly discussed and resolved.

1. We have updated the system's technical environment to Java instead of C++.
2. We made the Data Architecture more specific and refined.
3. All Cruise Control System Architecture Diagrams now have a descriptive summary.
    a. Architecture Context Diagram
    b. Cruise Control System Archetype
    c. Cruise Control Top-Level Component Architecture
    d. Cruise Control Refined Component Architecture
4. The UML Class Diagram now has a clear description of the relationship between the classes.
5. The requirement for sending requests to sensors before activation has been added.
6. The requirement for the power supply of the cruise control system has been added.
7. We have all agreed upon a complete list of classes for the Cruise Control System to operate properly.
8. We have clearly defined the order/priority of each object in the event to prevent errors with the transfer of data among the classes.
9. The components, connectors, and constraints have been distinguished from each other.

FTR Review Summary:

Total errors found: 14 minor errors + 2 major errors = **16 total errors**

Error Density: 16 total errors/30 pages = **0.53 errors/page**

Review Outcome:

- Participants in favor of accepting the Cruise Control Software Document provisionally: Ryan Qin, Jeffrey Lee, Christopher Moon, Siddhanth Patel