Cruise Control Software Development Version 0.07

We pledge our honor that we have abided by the Stevens Honor System.

CS347: Software Development Processes | Spring 2020

Team Mike

Eric Altenburg
ealtenbu@stevens.edu

Michael McCreesh
mmccree1@stevens.edu

Hamzah Nizami
hnizami1@stevens.edu

Constance Xu
cxu16@stevens.edu

Contents

1 Executive Summary

Team Mike is a startup initiative aimed at solving problems that come to light as society begins to adopt new technologies. One of which pertains to autonomous driving and "smart cars" as they are beginning to break into the automobile market more and more every year with examples such as Tesla and Ford. In a perfect world, if every driving car were to be a smart car with "autopilot," then it would make sense for each of them to communicate cruise control data with each other to reduce traffic build-up and make traveling more efficient. Through rotational leadership on a monthly basis, each team member possesses a set of distinct skills that can translate to highly efficient work sessions. Aside from developing a traditional cruise control, the aim is to make the software open source as it will serve as the foundation to a more interconnected logistical future in which autonomous cars can fully achieve their potential within a growing technologically advanced society.

2 Introduction

The past century has seen an explosion of innovation on a scale never before seen in human history. The rapid development and refinement of a myriad of motor technologies catalyzed the innovation process by allowing ideas to transfer at rates never before seen. With the constraint of distance loosening thanks to every iteration of the automobile, people have been granted more freedom to do what they please. While the automobile has enhanced society in several ways, there are some glaring problems that need to be addressed to continue the current rate of human progress and innovation. One of the biggest problems is driver fatigue, which is responsible for approximately 72,000 crashes annually. Programming the automobile to be reliably autonomous to a degree is one way to circumvent the issue of driver fatigue and making roads safer. That's exactly what cruise control aims to do. By moderating the speed of the vehicle by itself, cruise controls aim to lessen the effects of driver fatigue on the road. However, the technology is not perfect, and Team Mike aims to resolve that by creating the perfect cruise control system.

When it was implemented in 1958, cruise control was suitable for that era, however, as technology continually improves, the embedded software must as well. For longer car trips, it does not make sense for users to constantly accelerate and decelerate for several miles; this is where cruise control comes in. By being able to set the speed, the user is now able to accurately set and maintain the speed they wish to without having to constantly intervene which, over time, would cause less harm to the vehicle as variable speed and RPMs are not as desirable as that of near-constant.

Our group intends to use an Agile method to implement our version of cruise control. This would encourage code sprints in two-week lengths where work will be divided into smaller sections and distributed based on each members' strengths. The Agile method may vary as to which one we specifically choose but regardless, this project will not come to fruition using Waterfall or other methods. The way that we are going to organize our group is through weekly meetings and ensuring that everyone knows their tasks for the week. These weekly meetings can also serve as a place where team members mention any obstacles that have come in their way when trying to resolve a problem, and also serve as a time where the team as a collective can try to think of solutions around those obstacles. We also plan on giving real-time updates about any progress made through Slack in order to ensure all members of the team are equipped with the most recent developments of the project. We also intend on using Java or C++ to

implement our cruise control system. We believe that the high-performance Java provides and the fact that its part of the Object-Oriented Programming (OOP) paradigm makes it perfect for embedded systems.

There are a plethora of features that make up cruise control. As most cruise controls are seen on vehicles, there are generally a few buttons: increase speed (and sometimes decrease speed) and the button that starts cruise control. Once cruise control is set, the user then has the ability to increase, decrease, or maintain the speed of a vehicle. Furthermore, if the user presses on the brake, then cruise control is automatically deactivated. The system also allows for the operator of a vehicle to manually deactivate it.

To successfully implement the features of maintaining a steady speed and a safe distance away from other cars, there are certain requirements to fulfill. At a high level, proper hardware with fast, reliable sensors is critical for this project. Furthermore, lives are fundamentally dependent on this product and therefore it requires software that is error-resistant and thus a strong development and QA (quality assurance) team. In addition, the software must also perform it's expected tasks of moderating speed, maintaining a safe distance and switching back control to the driver when prompted to quickly. Having a cruise control software that is slow would be ineffective, so speed and accuracy are critical requirements for the software as well as the hardware. This is a mission-critical system because if it were to fail, it would put the lives of people in danger.

3 Requirements

3.1 Input

1. The system shall accept electric power from the alternator.
2. The power button that allows for the state of the system to change from on to off or vice versa.
3. When pressed, a button will either accelerate or decelerate in 1 mph increments.
4. When the brake is pressed, the cruise control system will unset the speed and give control back to the user until the user specifies a new speed to be set.
5. If the entire car turns off, then a signal will be sent to turn off the cruise control system.
6. RPM (Rotations Per Minute) sensor should be connected to the front axle and able to take readings for accurate speed calculations.
7. Engine sensor must be able to take input from the engine to tell the cruise control system to turn on or off.
   (a) If the engine is turned on, the cruise control system must be ready for use within 3 seconds of the engine being turned on.
8. If the gas pedal is pressed, the vehicle will continue to accelerate at the control of the user, but when the gas pedal is released the cruise control system will continue to the previously set speed.

3.2 Output

1. Keep a log of activity in the system files to help debug in the event of a malfunction.
2. For every speed increase made by the user in the cruise control system, a visual indication by the software is necessary. This is so that the user minimizes their own human error. So, every time you increase the speed, you can see the cruise control speed on the car menu increasing by however much the user wants it to.
3. The system displays successful activation or deactivation in 10 milliseconds.

(a) Numbers are subject to change depending on how inputs are received from surrounding system, but are expected to be in around that same ballpark.

4. Keep a log of every time the cruise control system is used and at what speed written to the system files.

5. Once the cruise control system is turned on, it must be readily available. The engine sensor must be able to understand that the engine is on and tell the cruise control system that, if the user so wishes, it must activate.

6. With all the inputs it is taking from all the sensors, the software must be able to deliver the desired output for each function in less than 15 milliseconds.

   (a) The brake pedal sensor must be able to tell the software that the user has stopped, and stop the cruise control system. *Numbers are subject to change depending on how inputs are received from surrounding system, but are expected to be in around that same ballpark.*

3.3 Functional

1. The system shall accept direct current from the car battery to support logging after engine shut down.

2. The system shall receive the time and date from the car's clock every second.

3. A physical will be provided by the system for technicians to access the unit.

4. Hardware shall have a 4 nine (99.99%) availability.

5. Software shall have a 5 nine (99.999%) availability.

6. While the cruise control system is on and a speed is set, it must be able to increase and decrease the speed by 1 mph.

7. Able to switch the cruise control system on or off provided the engine is on.

8. Interpret engine on and off signal to allow for the cruise control system to be turned on within 3 seconds of received the signal.

9. Only allow the cruise control system to be activated when at a minimum speed of 25 mph.

    (a) If the speed is set to 25 mph, the user cannot decrease the speed below 25 mph.

10. Maximum speed that cruise control system can be set to is at 125 mph.

11. While the cruise control system is on, if a user is driving at a speed of at least 25 mph and decides to set the speed, the cruise control system will maintain that speed.

12. If the user presses and continues to press the brakes, the user will not be able to set the speed.

13. If the user presses and continues to press the gas pedal, the user can set the current speed for the cruise control system, but it will immediately pause as stated in the input requirements then resume after the user releases the gas pedal.

3.4 Security

1. No external interface to reduce potential tampering. This applies to both the hardware such as sensors and the general cruise control system software. There would be no easy access to the cruise control system hardware or software so that the likelihood of someone being able to create issues is reduced substantially.

2. No Internet or Bluetooth connection. This is so that no bad actors can meddle with the car's system and hence, keeps the drivers safer as cyber security becomes more of an issue.

3. An administrator will need hardware to make changes to the mechanisms. The administrator must be authorized to make these changes and they must be a trusted third party or those who created the cruise control system themselves.

4 Requirements Analysis Model

4.1 UML Use Cases & Diagram

See Figure 1 for general use cases of cruise control system.

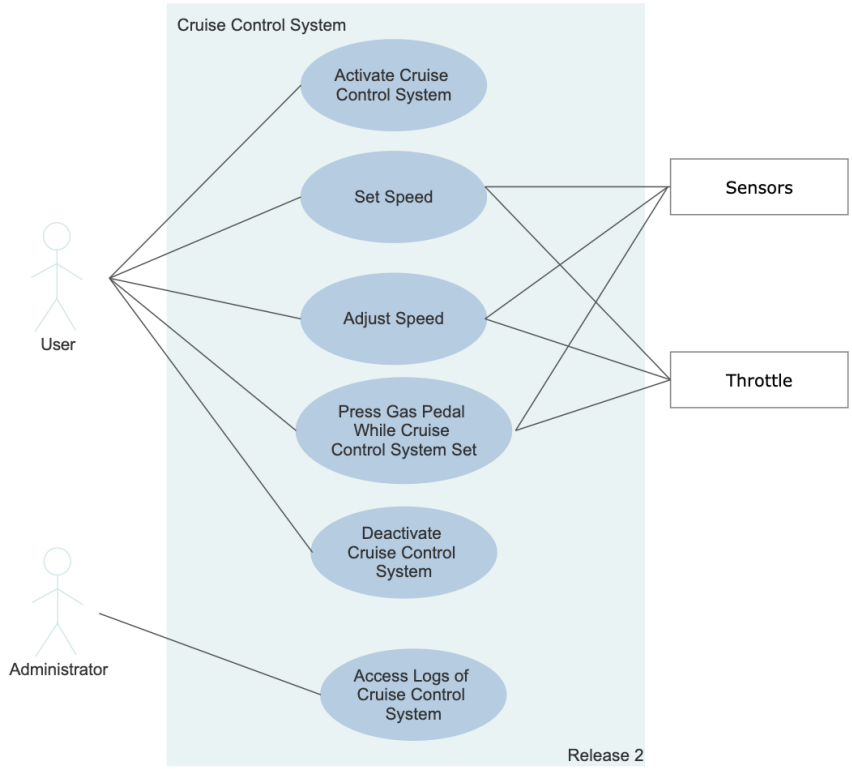

Figure 1: Sample UML diagram for cruise control system use cases.

It should be mentioned that there are two similar sounding states that need to be clarified:

- Activated
  - Granted the vehicle is on, the cruise control system is also now waiting for the speed to be set by the user.
- Set/Unset

- Given that the cruise control system has been **activated**, the user has requested for the speed to be maintained at its current speed.
- Given that the cruise control system has been **activated** and the cruise control system is in the **set** state, the user can either press the brake or press a button to unset the speed and revert back to the **activated** state.

1. Use Case 1: User activates the cruise control system
   (a) Given that the vehicle is on, the user requests an activation of the cruise control system.
   (b) Cruise control system activates.
   (c) The cruise control system provides a visual feedback that it has been activated.
2. Use Case 2: User sets the speed
   (a) Given that the vehicle is on and the cruise control system is in the activated state, the system requests values from sensors.
   (b) Sensors provide values for approval to set cruise control system at current speed.
   (c) Cruise control system requests the Engine Management System (EMS) set the speed at current position.
   (d) EMS Speed (Throttle) is set at current speed.
   (e) Cruise control system provides visual feedback to the user that the cruise control is set and working.
   (f) Sensors provide the changing environmental information to the cruise control unit (such as speed, request for increase/decrease speed, and brake).
   (g) Cruise control system detects the changes from the sensors and request adjusting speed or deactivating Cruise Control system accordingly.
   (h) Speed (Throttle) position is continuously set to new values to ensure that the speed remains constant.
   (i) Speed is continuously reported to the cruise control system.
3. Use Case 3: User adjusts speed

(a) If the cruise control system is in the set state, the user requests an adjustment of cruise control system speed.

(b) Cruise control system provides visual feedback that the system will alter the speed of the vehicle.

(c) Cruise control system slowly adjusts the speed of the vehicle to match that of the request.

(d) When the desired speed is reached, the cruise control system will provide visual feedback that the adjustment has been completed.

4. Use Case 4: User presses gas pedal while the cruise control system is set

   (a) If the cruise control system is in the set state, the user presses on the gas pedal.

   (b) In the event that the vehicle accelerates, the cruise control system will continually request the EMS to be set to the previously specified speed.

   (c) Speed will be continuously reported to the cruise control system.

   (d) After the user releases the gas pedal and the vehicle stops accelerating, the cruise control system will request for the EMS to be set to the previous specified speed.

   (e) The vehicle will naturally decelerate nearing the old set speed, and once reached, the cruise control system will resume with the set speed.

5. Use Case 5: User deactivates cruise control system

   (a) If the cruise control system is activated, the user requests a deactivation of the cruise control system.

   (b) Cruise control system deactivates.

   (c) Cruise control system provides visual feedback that it has been deactivated.

6. Use Case 6: Technician accesses logs of cruise control system

   (a) With the vehicle turned on and the cruise control activated, an administrator will have to use a proprietary physical hardware key in order to gain root access.

   (b) Once access has been granted, the administrator can download the logs to an external storage device through a USB port.

(c) Once downloaded, the administrator must log out using the same proprietary physical hardware key.

4.2 UML Class-Based Modeling

1. See Figure 2 for the class-based modeling for the cruise control system.

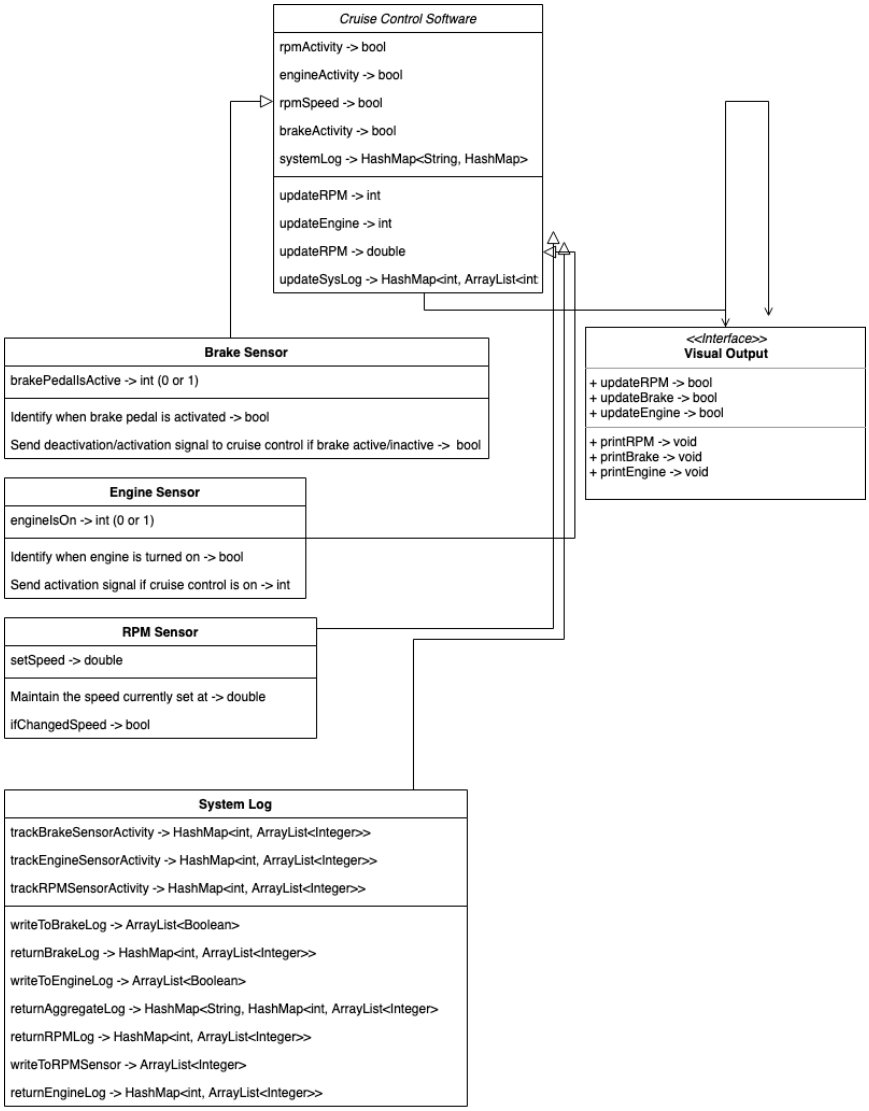

Figure 2: Sample UML class-based model for the cruise control system.

4.3 UML CRC Model Index Card

1. See Figure 3 for the CRC Model Index Cards for the cruise control system.

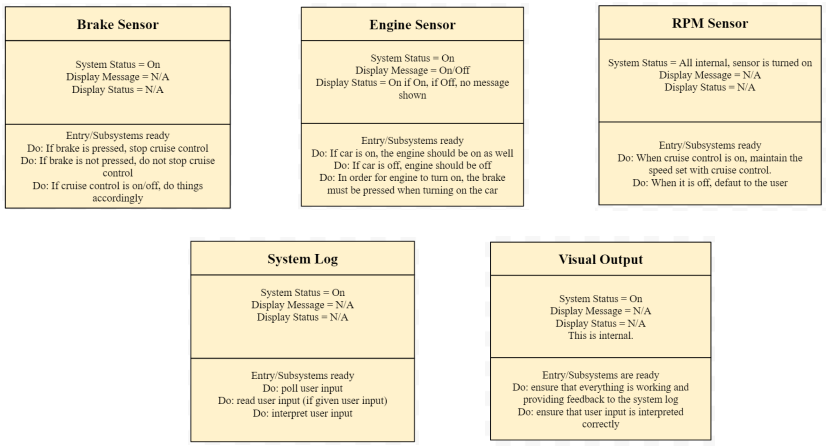

Figure 3: Sample UML CRC model index cards for the cruise control system.

4.4 UML Activity Diagram

1. See Figure 4 for the activity diagram of the cruise control system.

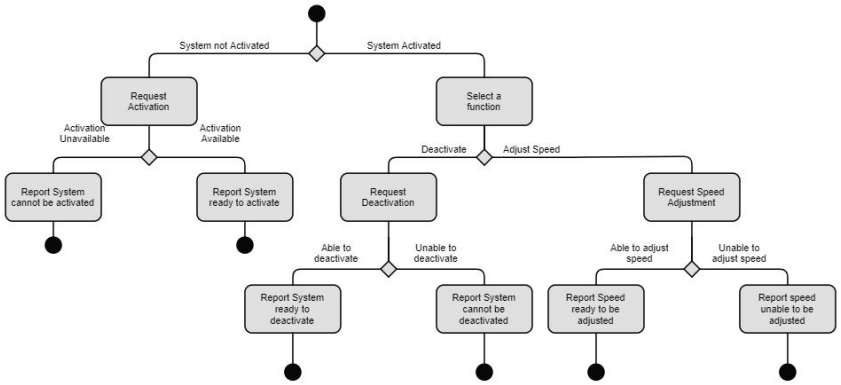

Figure 4: Sample UML activity diagram for the cruise control system.

4.5 UML Sequence Diagram

1. See Figure 5 for sequence diagram of the activation of the cruise control system.

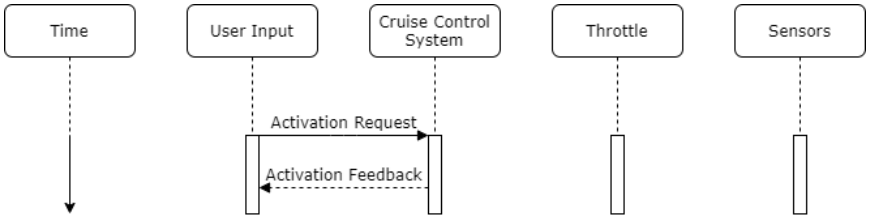

Figure 5: Sequence UML digram for activation of the cruise control system.

2. See Figure 6 for sequence diagram of cruise control system setting speed.

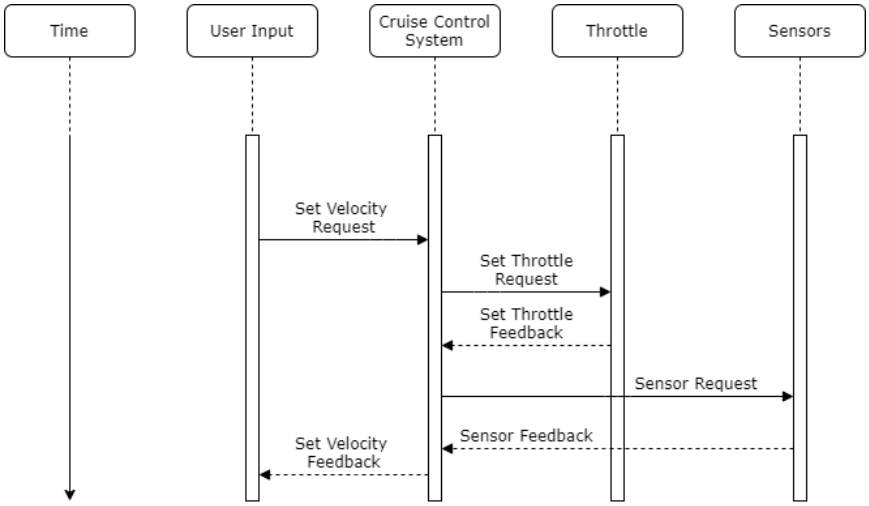

Figure 6: Sequence UML diagram for the cruise control system setting speed.

3. See Figure 7 for sequence diagram of cruise control system adjusting speed.

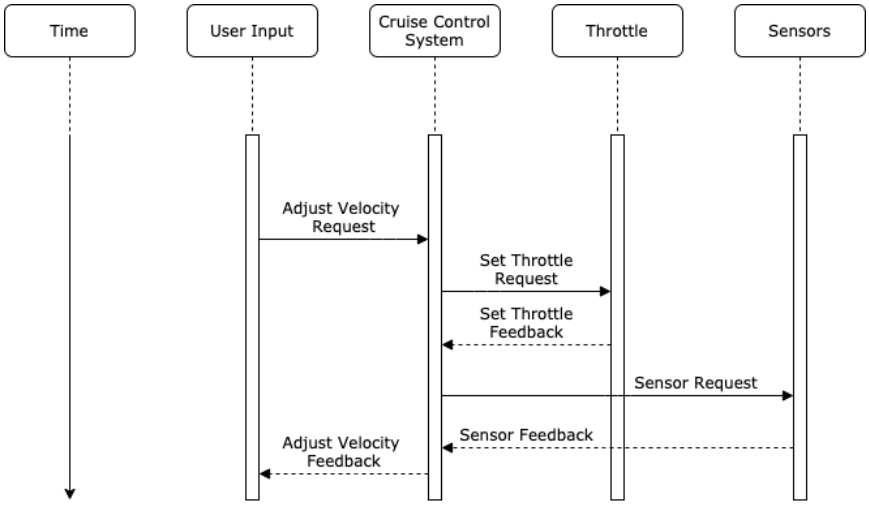

Figure 7: Sequence UML diagram for the cruise control system adjusting speed.

4. See Figure 8 for sequence diagram of the cruise control system being suspended when the user presses the gas pedal.

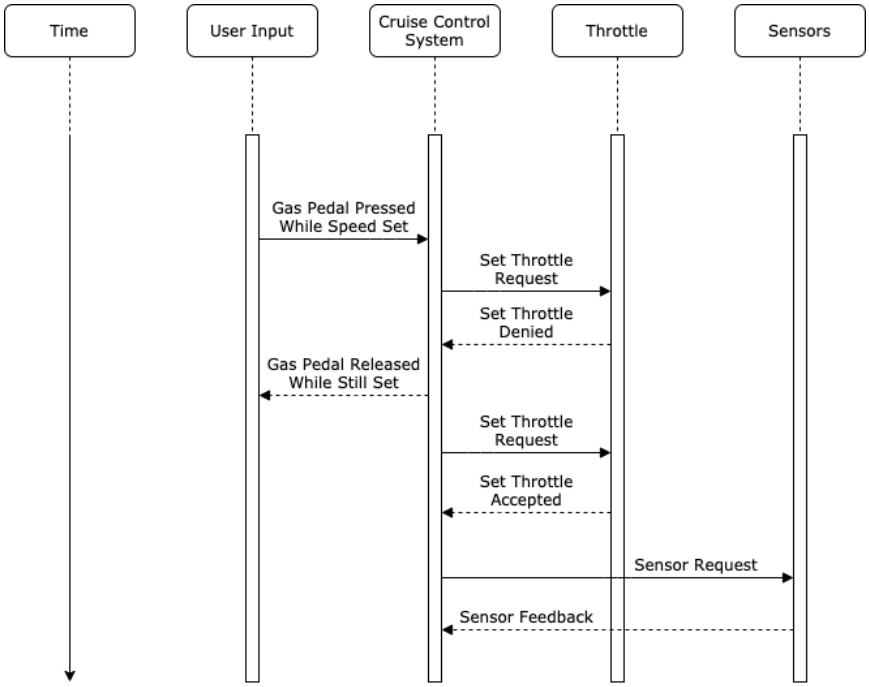

Figure 8: Sequence UML diagram for the cruise control system suspending during gas pedal being pressed.

5. See Figure 9 for sequence diagram of the cruise control system being deactivated.

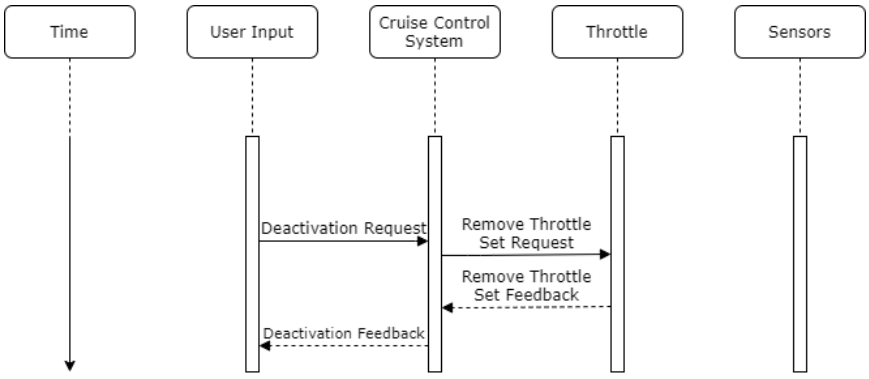

Figure 9: Sequence UML diagram for the cruise control system being deactivated.

6. See Figure 10 for sequence diagram of the administrator accessing the cruise control system logs.

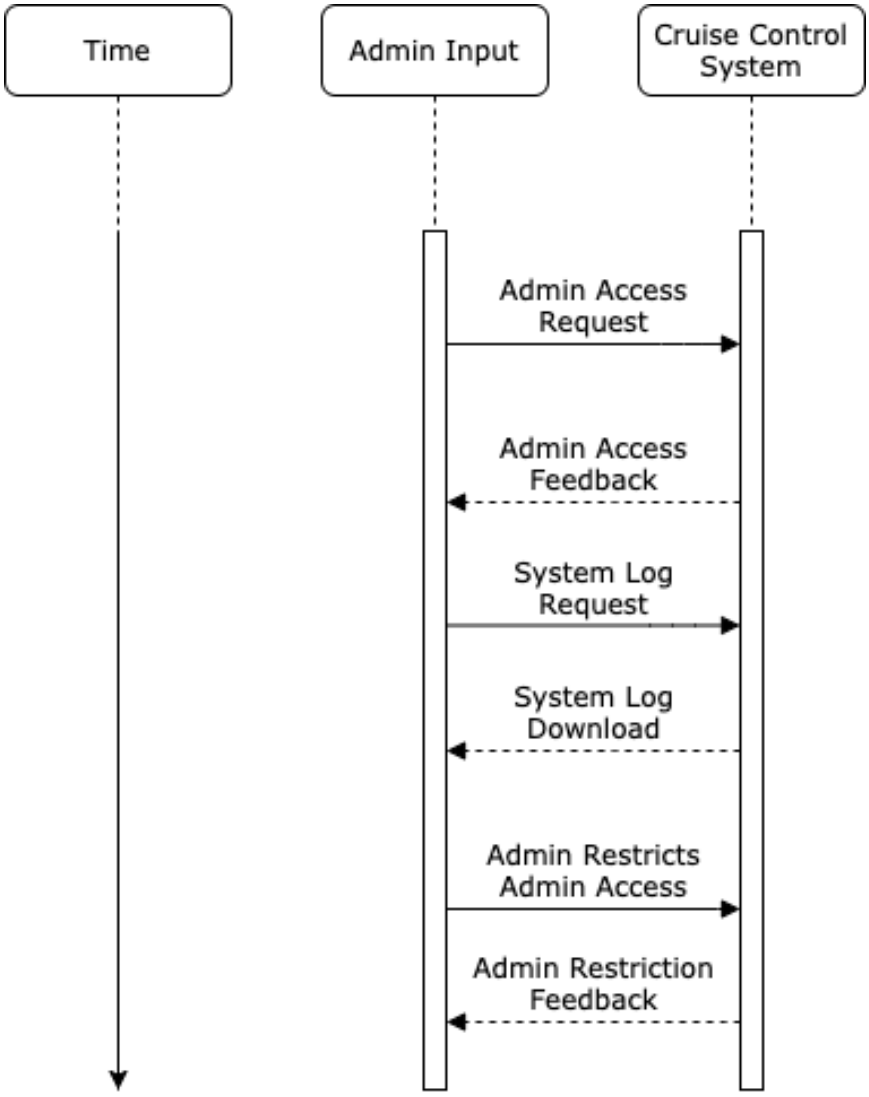

Figure 10: Sequence UML diagram for the administrator accessing cruise control system logs.

4.6 UML State Diagram

1. See Figure 11 for the state diagram of the cruise control system.

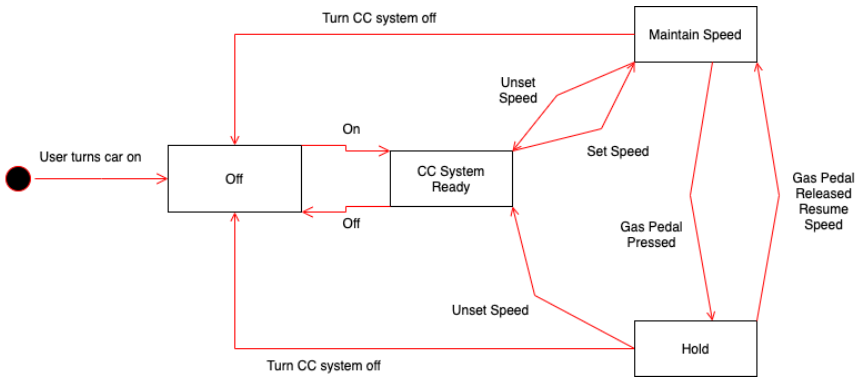

Figure 11: Sample UML state diagram for the cruise control system.

5 Software Architecture

5.1 Architecture Style

We will be implementing a combination of the object oriented architecture and the call and return architecture.

1. Call and Return
   (a) Pros: Easy to modify and scale. Grow system functionality by adding more modules. Simple to analyze and control flow.
   (b) Cons: Parallel processing may be difficult. Difficult to distribute across machines. Exceptions to normal operations are awkward to handle.
   (c) This makes sense with this project due to the many actions that can be performed by the cruise control system, for example acceleration, deceleration, and maintaining speed.
   (d) Different user actions should invoke different responses from the cruise control system. These responses are the result of activating different programs.
   (e) Specifically the type of call and return architecture we will be using is the main program/subprogram architecture.
2. Object-oriented
   (a) The components of a system encapsulate data and the operations that must be applied to manipulate the data. Communication

and coordination between components are accomplished via message passing.

(b) Pros: It models the real world very well. With OOP, programs are easy to understand and maintain. OOP offers code reusability. Already created classes can be reused without having to write them again.OOP facilitates the quick development of programs where parallel development of classes is possible.With OOP, programs are easier to test, manage and debug.

(c) Cons: With OOP, classes sometimes tend to be over-generalized. The relations among classes become superficial at times. The OOP design is tricky and requires appropriate knowledge. Also, one needs to do proper planning and design for OOP programming. To program with OOP, the programmer needs proper skills such as that of design, programming and thinking in terms of objects and classes etc.

5.2 Components

A software component is an architectural entity that encapsulates a subset of the system's functionality and/or data. A set of components will perform a function that is required by a system. This restricts access to that subset via an explicitly defined interface. The components of a software system are the following (but not limited to):

1. Network and internet services
2. Hardware level of operating system
3. Logical level of operating system
4. Graphics engine
5. User interface
6. System services
7. Command shell
8. System utilities

A software connector is an architectural entity affecting and regulating the interactions. This is what interacts among other components and has the

rules that govern those interactions. These interactions can be simple interactions such as procedure calls and shared variable access. Some complex interactions are client-server protocols, database access protocols, and asynchronous event multicast. Each of these connectors provides interaction ducts and transfer of control and/or data. Connectors in software system implementations usually have no dedicated code and no identity. They typically do not correspond to compilation units and have a distributed implementation across multiple modules and across interaction mechanisms.

Components and connectors are used to accomplish a system's goal. This is expressed through an architectural configuration. More precisely, an architectural configuration is an association between components and connectors of software architecture. A connector is not equivalent to a component as components provide application-specific functionality while connectors provide application-independent interaction mechanisms.

A software constraint defines how components can be integrated to form the system. A software constraint is a restriction on the degree of freedom you have in providing a solution. Constraints are effectively global requirements, such as limited development resources or a decision by senior management that restricts the way you develop a system. Constraints can be economic, political, technical, or environmental and pertain to your project resources, schedule, target environment, or to the system itself. Some constraints are the following:

1. The system will work on our existing technical infrastructure in which no new technologies will be introduced.

2. The system will only use the data contained in the existing corporate databse.

3. A programming language can also be a constrain because oftentimes, a specific programming language will be required for various reasons.

5.3 Control Management

Call-and-return uses a looping structure to control environment variables. It uses a pipe -filter system which creates an incremental transformation of streams. Pipe-filter can be used to structure systems that produce and process a stream of data. Each processing step is enclosed within a filter component and data to be processed is passed through pipes. These pipes can be used for buffering or for synchronization purposes.

Call-and-return is split so that there is a main program that is divided into smaller pieces hierarchically. The main program invokes many program components in the hierarchy that program components are divided into subprograms. Hence, the main program is the driver of a call-and-return architecture. The main program or subprogram components are distributed within a network of multiple computers. The main aim is for an increase in performance. Call-and-return is used in object-oriented architectures which consists of bundling data and methods. For our purposes, the call-and-return main function would control incrementing the speed of a car or decrementing it and is responsible for the activation of cruise control. Call-and-return incurs a finite communication time between subroutine call and response. It strives to increase performance by distributing the computations and taking advantage of multiple processors.

5.4 Data Architecture

In terms of data architecture, the plan is to adopt a scalable, object-oriented, NoSQL database like MongoDB. Considering that the project is going to be programmed in an object oriented language such as Java or C++, having the data schema represented in the same programming paradigm streamlines the technical thought process behind the project. Furthermore, MongoDB is known for its high availability and performance. Having a data schema known for reliability is imperative for a mission critical piece of software like cruise control.

Due to the use of Java/C++ and MongoDB, the general paradigm of the data architecture is object-oriented. We will also be using a call and return architecture in conjunction, which will be discussed later on in the document. An object-oriented data architecture means that the components of a system encapsulate data and the operations that must be applied to manipulate the data. Communication between the components is done via message passing. Due to the object-oriented nature of the system, that means that the data between components is communicated via synchronous message-passing. Message passing is a critical since it relies on all the components to be active in order to pass information. This makes it easier to identify when the system is down. This also means that data objects are passed around the system based on need.

The mode of data transfer is direct memory access. Since data components exist in this cruise control (such as a data component for RPM sensors, one for brake sensors, one for engine sensors, one for system logs), each component can access each other if they need. So, if a system log needs to be

generated, it can access RPM sensors, brake sensors and engine sensors data directly and then compile and create the log files. Thanks to this direct access model, functional components itneract with data components directly as well. For example, the visual interface can simply query the RPM sensor object to obtain the information needed about speed. There are no proxies or extra steps needed, thus avoiding the chance of dependency issues. This is where the call-and-return architecture comes in. If a functional component needs to access something, a subprocess can run to access the needed data without detering the general flow more mission-critical pieces of the software.

As we can see, the control of the program sets up the relationship between itself and the data on a need by need basis. Whenever the program needs something, it can directly access the data object to store, modify or delete values.

5.5 Architectural Designs

5.6 Issues