Software Requirements Specification (SRS)

Project Scalable Cruise Control #2

Authors: Ian Murray, Kira Chan, Brandon Brooks, Prudhvi Kuchipudi, Zebin Liang

Customer: Ford Motor Company

Instructor: Dr. James Daly

1 Introduction

The software requirements specification (SRS) document outlines the technical requirements needed by system developers to implement the scalable cruise control (SCC) system in eligible Ford automobiles.

Section 1 (Introduction) provides a general overview of both the SRS document and SCC solution. The purpose section illustrates the purpose and intended audience of the SRS document. The scope section identifies and briefly describes the benefits, objectives, goals, domain, and high-level functionality of the scalable cruise control system. Throughout the SRS, several abbreviations, acronyms, and technical terms are used for the sake of brevity. Not all these terms are inherently understood to the reader, so the definitions, acronyms, and abbreviations section defines each of these terms in greater detail. Finally, the organization section provides a table of contents regarding the remainder of the SRS document.

Section 2 (Overall Description) outlines the scalable cruise control system in greater detail than section 1. It begins with a product perspective, which illustrates the context, scope, and constraints of SCC. The product functions are also given, which summarizes the major functions of the software from a user's perspective. The user characteristics section provides developers with assumptions regarding the user's background, skill level, and general expertise with similar systems. The constraints sections list restrictions based on safety and mission-critical operations. A list of hardware, software, environment, and UI assumptions and dependencies are provided after that, and finally a list of customer stretch goals and future release requirements are provided in the approportioning of requirements section.

Section 3 (Specific Requirements) provides an exhaustive and hierarchical list of technical requirements for the scalable cruising system. It covers all aspects of the SCC. Unlike sections 1 and 2, this section is intended for development, instead of overview purposes.

Section 4 (Modeling Requirements) visually demonstrates much of the implementation strategy of the scalable cruise system. It bridges the gap between the application domain and the machine domain. Section 4 includes several use case diagrams, which illustrate the user and system interaction with the system to accomplish a goal. Section 4 also includes a high-level class diagram, which depict the key elements of the system. This class diagram includes a data dictionary to describe each class, along with their attributes, operations, and relationship with other classes. Following the class diagram is the representative scenarios section, which includes scenarios for each use case, along with a respective sequence diagram. Finally, a state diagram is included, which shows how each of the classes listed in the class diagram participate in the scenarios.

Section 5 (Prototype) outlines the requirements and the intended functionality (shown through sample scenarios) of the first SCC prototype. The requirements listed include system configuration, necessary libraries, constraints regarding operating system and networking, along with an accessible URL for the prototype. The sample scenarios utilize real data, real scenarios, and screen captures to illustrate the prototype's functionality.

Section 6 (References) lists all the supplementary material referenced in the SRS document. It includes the title, report number, data, and publisher, along with the source for each reference. The developer's website can also be accessed in this section.

Section 7 (Point of Contact) provides contact information for those inquiring for more information regarding the SRS or SCC.

1.1 Purpose

The software requirements specification (SRS) functions as an exhaustive reference document intended for developers implementing the scalable cruise control (SCC) system for the Ford Motor Company. The SRS document is useful for developers familiarizing themselves with the project through the summaries provided in sections 1 and 2. These sections also allow developers to maintain overall focus on the core of the project. The SRS is also useful for the actual implementation of SCC, through the technical details and diagrams provided in sections 3, 4, and 5. Finally, the SRS allows developers to find additional information if desired, through the references and contacts listed in sections 6 and 7.

1.2 Scope

The product sponsored by Ford is the scalable cruise control (SCC) system. This system aims to enhance the versatility of existing cruise control technology by introducing a greater degree of automation into the cruising experience (SCC Requirements Definition, 2018). A successful implementation of SCC improves the safety and convenience of the driving experience, while also innovating on the fronts of automotive cybersecurity and following distance infrared technology.

The implementation of the SCC is entirely embedded in the domain of automotive computing and dashboard systems. The system is mostly backend, with calculations done inside the vehicle's computer systems. A minor amount of user interface is also required on the dashboard, as the motorist must be able to send commands to the system, along with seeing speed and distance-based output.

Scalable Cruise Control enhances the driving experience in several ways. The most prevalent benefits pertain to driver safety. In vehicles supporting only simple cruise, there are many use cases in which activating cruise control is unsafe. For example, cruise control is dangerous when the motorist is driving on any roadway with moderate to heavy congestion. In this scenario brake-checking, traffic snakes, and lane changes often occur unannounced. For an inattentive driver using only simple cruise, a collision will occur without manual intervention. However, with the following distance management module and automatic emergency brake module in place, the vehicle will know when to stop, and execute the stop long before a collision occurs (SCC Requirements Definition, 2018).

These safety benefits also affect user convenience. With the ability to automatically scale cruising speed, drivers do not need to hassle about manually entering minor changes in speed in relation to other vehicles. Instead, they are free to focus on the other aspects of driving. Furthermore, because the scalable cruise is based on *following distance* and not only speed, each respective driver can cruise to the trailing distance in which they are comfortable and confident driving in (SCC Requirements Definition, 2018*).*

The SCC is not intended to address all consumer complaints regarding existing cruise control. Resource constraints prevent this. The SCC is not to be retroactively fitted into older Ford vehicles. Such models lack the available hardware and drivers to safely implement SCC. Furthermore, the risk of implementing aftermarket technology is not one that Ford is willing to undergo. Furthermore, because the SCC is built atop existing simple cruise technology, the SCC does not change the way basic cruise works. If the driver wants to forego scalable cruise functionality, they will experience the same cruising experience as before, from both a front and backend perspective.

1.3 Definitions, Acronyms, and Abbreviations

- **SCC (Scalable Cruise Control):** Ford's proposed solution. SCC is built upon existing cruise control systems. Consists of three facets: simple cruise, following distance management, and automatic emergency brake.
- **SCC Module:** One of the independently acting facets of the SCC system (e.g. Simple cruise, FDM, AEB).
- **Simple Cruise:** The cruise control technology that exists in vehicles prior to SCC integration.
- **FDM (Following Distance Management):** An SCC module that allows motorists to customize their cruising experience based on distance to the preceding car, rather than a constant speed.
- **AEB (Automatic Emergency Brake):** An SCC module that takes control of a vehicle's brakes if the driver is unable to manually prevent an impending collision. This module is active regardless of cruise control's status.
- **Intrusion Prevention System (IPS):** A technology that examines traffic flows to and from critical systems to detect and prevent cybersecurity breaches.
- **Internet of Things (IOT):** The ever-increasing integration of internet access into everyday objects such as appliances and vehicles.
- **National Highway Traffic Safety Administration (NHTSA):** The US-based regulatory board that issues laws and restrictions on vehicle manufacturers.
- **Original Equipment Manufacturer (OEM):** The brand and model of part that was included with the vehicle in its initial assembly.
- **Cruising Speed:** A user-defined speed which the vehicle will try and maintain.
- **Cruising Distance:** A user-defined distance which the vehicle will try and remain behind the object in front of the vehicle.
- **Cruise Enable:** To activate the cruise control system's features (i.e. turn on).
- **Cruise Disable:** To deactivate the cruise control system's features (i.e. turn off).
- **Cruise Suspend:** To temporarily pause the cruise control system's features but save its current settings.

- **Cruise Resume:** To return the cruise control to its full activation state.
- **Maximum Possible Safe Speed:** The highest speed the vehicle can maintain while still avoiding the possibility of a rear-end collision with the leading vehicle.
- **UI (User Interface):** The abstracted portion of the SCC that is presented directly to the vehicle operator.
- **I/O (Input/Output):** The potion of the SCC that allows for communication between the operator and the SCC system itself.
- **ACK:** Short for acknowledge. A small signal a receiving component replies to a sending component that important data was received and correctly formatted.
- **Minimum Time to Intercept (MTTI):** Used by the FDM to determine the correct amount of distance to stay behind the preceding vehicle.

1.4 Organization

The remainder of the SRS includes both summary and detailed requirements regarding the SCC implementation. It also includes references and contact information for supplementary resources. See table of contents to locate a specific section.

2 Overall Description

This section introduces the SCC with respect to its context, functionality, assumptions, and constraints. It also contextualizes the user, with respect to their domain experience and expectations of stretch goals. This section is intended for design-level specification. Implementation-level specifics are found later in the SRS.

2.1 Product Perspective

The context of the scalable cruise control system is mostly in the vehicle electronics - implemented by appending new modules onto a vehicle's existing cruise control technology. Since the simple cruise already has access to the vehicle's driving mechanisms, accelerometer and brakes, the axillary FDM module can accomplish its duties by interacting only with simple cruise. As a result, FDM only needs access to the vehicle electronics. However, because AEB must function independently of cruise, it needs access to braking functionality. This decision is beneficial to both the developer and the endpoint consumer. Developers can save resources by working upon existing frameworks, so new drivers and I/O modules do not need to be developed. End users can also easily adapt to SCC, as it replicates cruise control UI's they had become accustomed to. Figure 1 illustrates how SCC is incorporated upon a vehicle's simple cruise module.

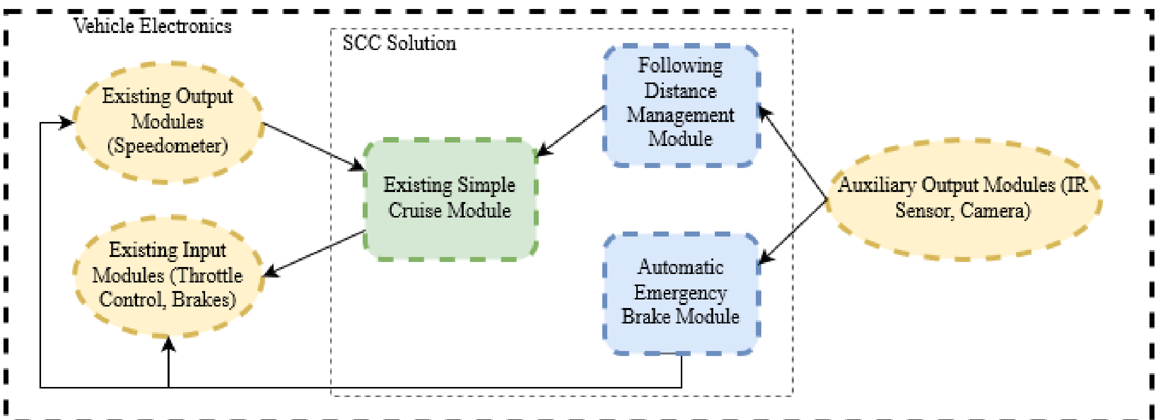

Figure 1: SCC Data Flow Diagram

The SCC system consists of a multi-faceted solution incorporating new and existing technology. The foundation of the project is the *simple cruise system*, the current cruising solution present in most vehicles. This is the basic speed control interface which most drivers use and are familiar with. For added functionality, a *following distance management* module grants drivers the ability to set their cruise to a trailing distance, rather than a constant speed. The third and final facet of the Scalable Cruise solution is the *automatic emergency brake.* This component, to be a standard feature beginning in 2022, takes control of the vehicle's brakes when the driver lacks sufficient time or strength to manually brake against an imminent collision.

The scalable cruising solution must function under numerous constraints. Regarding system interface, the SCC auxiliary modules (those built upon simple cruise), must interact directly with the simple cruise module. This is because the FDM needs access to speed, accelerometer, and brake I/O, and can only gain access through simple cruise. As a result, the simple cruise module must be able to receive cruising related commands from the advanced modules and pass them to the in-place input modules as an intermediary.

With respect to the user interface output, the primary constraint is the limited amount of space provided on an already-crowded vehicle dashboard. The SCC needs to present all relevant information, such as current speed/following distance and desired speed/following distance without impeding on other systems' output. This may include finding a way to reduce the amount of SCC output or finding other systems' output to suppress. Considering UI input, the SCC must similarly implement all functionality with a limited number of user-accessible buttons. The buttons must allow the operator to: enable and disable the cruise control system, suspend and resume the cruise control system, set the initial cruising speed, set the initial cruising distance, increase or decrease the cruising speed, and increase or decrease the cruising distance. (SCC Requirements Definition, 2018). The number of buttons may not match the number of required features, so these buttons may vary in functionality depending on which module(s) are currently active. The UI must also be convenient so that it is used by drivers. Confusing features are often not worth the hassle to operators, especially those who are not very familiar with driving or have been driving a very long time. Most drivers are familiar with the simple cruise interface. Due to this existing familiarity, additional cruising modules must match this UI as closely as possible. It is important for the SCC to be minimally intrusive to minimize the learning curve of adapting to vehicles with Scalable Cruise enabled (SCC Requirements Definition, 2018).

From a hardware perspective, the main constraint is that the hardware processors in the automobile be powerful enough to effectively compute the required analysis related to cruising. The SCC involves many simultaneously running calculations in a safety-critical environment, so latency issues are an unacceptable risk. Additionally, all hardware used must be Ford OEM parts, for security and reliability purposes.

The constraints involving SCC's software involve the solution being as lightweight as possible. A lightweight solution maximizes security of the system. A smaller codebase leaves less opportunities for vulnerabilities. This is important as if the SCC modules are vulnerable to remote attacks, then safety is no longer guaranteed, regardless of system performance (SCC Requirements Definition, 2018).

Operationally, the key constraint is with each function's time complexity. Since vehicles operate in real-time, so must the operations. Dangerous situations can arise for a motorist within a fraction of a second. As a result, all 3 SCC modules must utilize functions that can perform and return calculations as soon as the querying function needs them.

Memory constraints are not a major component of the scalable cruise control solution. Newer vehicles often have extensive computing environments capable of storing a great deal of logs and intermediary calculations. Since the SCC does not need to retrofit into older Ford automobiles, this is not a concern.

Site adaption operations constraints are also not of great concern to the SCC implementation. The end user interacts mostly with existing simple cruise technology, which has its customization abilities already in place. The only additional customization needed is the added ability for the user to switch from cruising speed to following distance. The only customization constraint is that this swap must be handled seamlessly by the other system components.

2.2 Product Functions

The simple cruise system functions as follows. The driver must be able to enable/disable cruising and set a maximum speed to be maintained. This speed is only saved if greater than 25mph. The vehicle must internally maintain this speed through throttle control and speedometer readings. The driver must be allowed to exceed this through accelerometer inputs or suspend it through a button press or brake pedal. Another button resumes the cruise to the last saved speed. Finally, the saved speed can be changed through button press (SCC Requirements Definition, 2018).

The following distance management system functions to the following customer specifications. The driver can set a following distance from a scale of 1 (close) to 4 (far) with button controls. This distance is internally maintained by the cruise controller based on the “time to intercept”, calculated by the position and relative speed of leading vehicle. The vehicle maintains the maximum possible safe speed through throttle and brake control, considering real-time inputs from itself and the following vehicle. These data points must come from the on-board camera and radar, within the user-set speed, distance, and manual throttle/brake constraints (SCC Requirements Definition, 2018).

Finally, the automatic emergency brake system must function to the following specifications. The AES will function regardless of cruising status. The AES, when prompted to take action, applies maximum braking force to minimize stopping distance. The AES takes action if it determines the driver cannot stop in time to avoid a collision depending on the vehicle speed, and camera and radar output (SCC Requirements Definition, 2018).

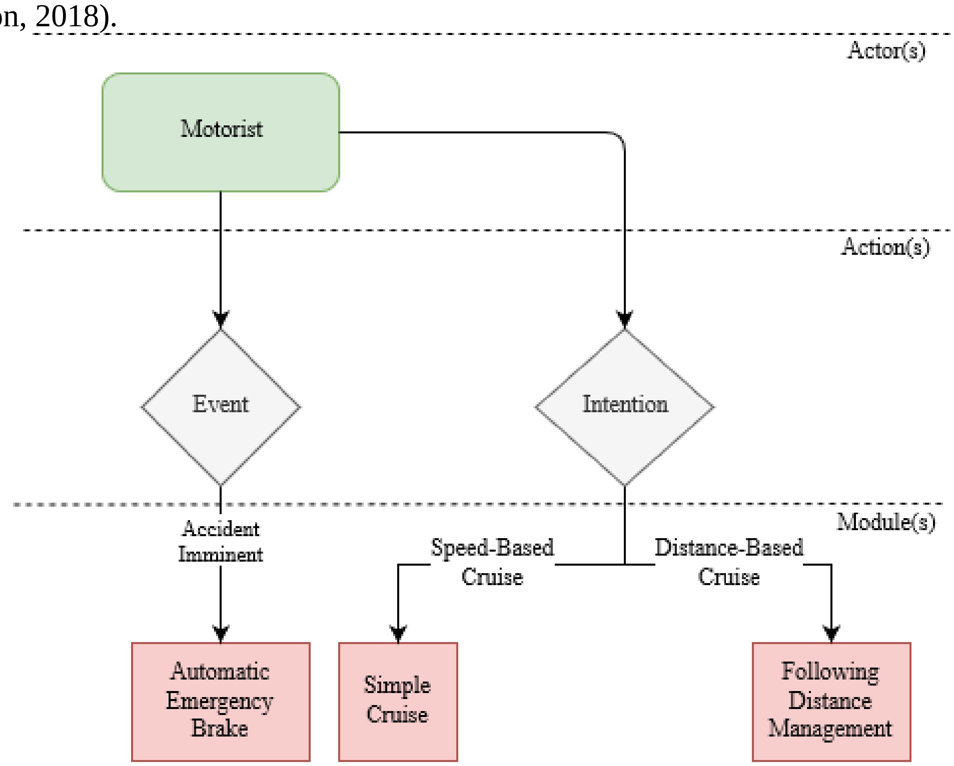

Figure 2: Goal Diagram
Green Box | Actor(s) Involves
Grey Box | Action(s) Invoked
Red Box | Module(s) Used

2.3 User Characteristics

The user of the scalable cruise control system is the average middle-class motorist. It is relatively safe to assume that young and working-class drivers are not willing to spend more on extra SCC functionality. It is also safe to assume that older, wealthier motorists value luxury brands and comfort over novel technology. Therefore, the ideal consumer for a Ford vehicle with SCC technology is a middle-class driver with 10-40 years of driving experience. This driver is open to learning and showing off new technology and may treat novel tech as a factor when buying a new automobile.

Since the projected customer base are middle-class adults willing to spend additional money on an SCC-supported vehicle, they likely have a reasonable amount of intuition and curiosity on learning how to use SCC. As mentioned before, these drivers also likely have a significant level of familiarity with existing simple cruise tech. This factor, combined with a descriptive UI and owner’s manual, means that presenting SCC to end users should not be a significant concern.

2.4 Constraints

The constraints of a successful SCC implementation are not solely based on internal design. The auxiliary cruising modules must also conform to all local, national, and international laws and regulations. If SCC functionality is prohibited in one or more jurisdictions, then the overall customer base is substantially limited. Of utmost importance is that the SCC conforms to the regulations set forth by the NHTSA and all other US Department of Transportation agencies.

As stated earlier, it is safety-critical that all functions operate as close to real-time as possible. This also applies to signal transferring between modules. If timing issues persist anywhere in the vehicle when SCC is in action, the system latency could result in injury or death to the vehicle operator.

All vehicles implement many more components than just cruise control. Motorists are often also able to manage components for radio, climate control, seat adjustment, towing, light control, and so on. Although many of these elements do not interact directly with cruise control, it is always possible that there exist unintended interferences. These systems must be able to operate seamlessly in parallel with one another. Even if the cruising module is foolproof on its own, if another vehicle component can accidentally interfere with it, then all guarantees of safety are lost.

The software constraints of SCC extend from the hardware. It is not always possible for a hardware component to report its own failure, so the software must be able to audit and report this for additional failsafe capacity. If any hardware component fails, the software must be able to make an executive decision to disable SCC until maintenance can be performed.

Cruise control functionality is not critical to vehicle operation. Although it is required that cruise works safely when activated, there is no impedance to the safe operation of the vehicle if cruise is disabled. Therefore, criticality is not a major concern. To remedy unsafe operation in case of a glitched SCC, the SCC activate function should be disabled until deemed safe by maintenance personnel.

The reliability requirements for SCC are very important. The SCC solution involves several sensor recorders including mission-critical ones that measure speed and distance. These components must not only be accurate, but fault-tolerant. Any deviation from the intended execution of any one sensor could plausibly fail the entire system. To better ensure reliability, signal handshake protocols must be implemented. A series of inter-component ACKs will suffice, to guarantee successfully communication of important data.

Cybersecurity is also a major concern to the SCC solution. The system must be protected by a firewall and other intrusion prevention countermeasures. The evolution of the Internet of Things (IOT) has brought forth a multitude to security vulnerabilities and exploits, many of which pose a danger to life and property in situations pertaining to vehicle operation. If a hacker gains access to a vehicle's accelerator or brakes, there is an unacceptable level of risk. Therefore, all unrecognized traffic must be blocked through relevant (WiFi, LTE, 802.11, etc.) channels via firewall (SCC Requirements Definition, 2018).

2.5 Assumptions and Dependencies

A successful SCC solution in based off several hardware assumptions and dependencies. First is the dependency that all the modules rely on only Ford automobiles and Ford-sponsored OEM parts. The SCC is built upon one uniform interface, so aftermarket components are unable to integrate with SCC. Ultimately this means that maintenance staff have to conform to OEM standards. Another assumption is that the relevant infrared distance sensors are be obstructed. If snow, dust, or other debris blocks the gap between the SCC and preceding vehicle, the distance marks as 0 and the SCC malfunctions. Finally, the SCC depends on all individual hardware components functioning properly.

Regarding software, again the SCC depends on the end user not installing aftermarket software, as their drivers may vary in syntax and interface. Each module also depends that there exists sufficient processing power and algorithm efficiency to receive data in near real-time. If this assumption is not true, latency issues hinder the system's functionality. Lastly, the software assumes that it is secure from cyberthreat intrusion. The SCC implements extensive intrusion prevention systems (IPS) but lacks in mitigation strategies. The system does not take any countermeasures to intrusion, as it assumes there are none.

Automobile accidents cause serious and often unintended consequences to the impacted vehicle components. The SCC assumes no physical damage has been dealt to its modules. If the vehicle tries to utilize SCC after a prior accident, its performance may be unstable.

The SCC only provide limited user input, so user interaction assumptions are not a major factor. Furthermore, any viable SCC solution should be resilient enough to handle any combination of user input. However, there are some edge cases that may cause issues. Automatic emergency brake assumes the user will not try to accelerate in its perceived emergency, and the SCC may fail in the event of conflicting manual acceleration and manual acceleration.

2.6 Apportioning of Requirements

There are several stretch goals of the SCC, to be implemented in a future release. The contact for this project, Dr. James Daly, has specified that future iterations (post-prototype) shall include the following features:

- **Cyber Threat Mitigation**: The prototype does not include any countermeasures to injection attacks, takeovers, or other security intrusions. This is an important requirement to the project, so it must eventually be implemented.
- **Automatic Emergency Brake Fine-Tuning:** Unlike simple cruise and following distance management, the specific variables regarding AEB are much harder to determine. Each Ford model varies in braking power. Furthermore, a vehicle's braking power also relies on the age of the brake pads. Due to these externalities, the prototype's implementation of AEB will be rough. During live field testing, these variables can be tweaked further.

3 Specific Requirements

This section outlines the list of requirements for each facet in the SCC implementation. The individual components are listed first, followed by the larger-scale modules that rely on these components. Level 1 of the hierarchy describes the module. Level 2 indicates the component in that module. Level 3 lists the requirements for that component.

1. Base Component Requirements
   1.1. Accelerometer
      1.1.1. **Accelerate Manual Start**: The accelerator will receive a request to increase speed. The accelerator module will ACK this request. The accelerometer module will open the fuel valve to allow fuel into the engine. The amount of fuel released will increase as more pressure is placed on the gas pedal.
      1.1.2. **Accelerate Manual Stop**: The accelerator will receive a request to stop increasing speed when no pressure is detected on the gas pedal. The accelerator module will ACK this request. The accelerator module will close the fuel valve to entirely stop fuel from entering the engine.
      1.1.3. **Accelerate Cruise Start**: The accelerator will receive a request to increase speed from the simple cruise module. The accelerator module will ACK this request. The request will specify a desired speed. The accelerator will open the fuel valve to allow fuel into the engine. The amount of fuel released will increase based on the difference between the desired speed and current speed.
      1.1.4. **Accelerate Cruise Stop:** The accelerator will receive a request to stop increasing speed from the simple cruise module. The accelerator module will ACK this request. The accelerator will close the fuel valve to stop fuel from entering the engine.
      1.1.5. **Accelerate Cruise Maintain**: The desired speed sent by the simple cruising module and current speed are the same, within a certain margin of error. The accelerator will keep the fuel valve open at its current level. This action will continue until it receives one of the previously listed overrides.
   1.2. Brakes
      1.2.1. **Brake Manual Start**: The brakes will receive a request to activate. The brake module will ACK this request. The brake module will apply

the brake pads. The pressure on the brake pads will increase as more pressure is placed on the brake pedal.

1.2.2. **Brake Manual Stop**: The brakes will receive a request to stop braking when no pressure is detected on the gas pedal. The brake module will ACK this request. The brake module will release all pressure on the brake pads.

1.2.3. **Brake Cruise Start**: The brakes will receive a request to decrease speed from the simple cruise module. The brake module will ACK this request. The request will specify a desired speed. The brakes will apply pressure to the brake pads. The amount of pressure will increase based on the difference between the current speed and desired speed.

1.2.4. **Brake Cruise Stop:** The brakes will receive a request to stop braking from the simple cruise module. The brake module will ACK this request. The accelerator will stop applying pressure to the brake pads.

1.3. Speedometer

1.3.1. **Maintain Speed:** The speedometer will internally maintain a record of the current speed of the vehicle.

1.3.2. **Return Speed:** The speedometer will receive a request to output the current speed of the vehicle. The speedometer will then return this record.

1.4. Infrared Distance Monitor

1.4.1. **Maintain Distance:** The distance monitor will internally maintain a record of the current distance between it and the closest obstacle in front of it.

1.4.2. **Return Distance:** The distance monitor will receive a request to output the current distance between itself and the preceding object. The distance monitor will then return this record.

1.5. Firewall

1.5.1. **Traffic Receive**: The firewall will receive packets from the intrusion prevent system. The firewall will ACK the reception to the IPS.

1.5.2. **Traffic Analyze**: The firewall will determine based on customizable heuristic models whether the traffic is benign or malicious in nature. If the traffic is benign, it will allow it. If it is malicious, it will deny the traffic.

1.5.3. **Traffic Allow**: The firewall receives a benign packet. The firewall will return that the packet was not malicious.

1.5.4. **Traffic Deny**: The firewall receives a malicious packet. The firewall will return that the packet was malicious.

2. User Interface

2.1. User Input

2.1.1. **Button Press:** This is the primary function of a button. The user intends to send a one-time signal input (e.g. increase speed, decrease following distance). The corresponding button will activate based on a pressure threshold. The button module will then send the activation signal to whichever module it is assigned to.

2.1.2. **Button Hold:** This is the auxiliary function of a button. The user intends to send a one-time signal input. The corresponding button will activate based on a pressure threshold and time-held threshold. The button module will then send the activation signal to whichever module it is assigned to.

2.1.3. **Button Latch:** The user intends to send a continuous signal input (e.g. cruise control ON). The corresponding button will activate and remain compressed based on a pressure threshold. The button module will send the activation signal to whichever module it is assigned to.

2.1.4. **Button Unlatch:** The user intends to cease sending a continuous signal input (e.g. cruise control OFF). The corresponding button will deactivate and remain uncompressed based on a pressure threshold. The button module will send the signal to whichever module it is assigned to.

2.2. Display Output

2.2.1. **Display:** The output module will receive a request to display data. The display module will ACK this request. The request will include what data to display, as well as where to display it. The output module will then modify the display to show the intended data.

3. Simple Cruise Requirements

3.1. **Cruise Enable**: The simple cruise module receives a request to activate simple cruise from the button input module. The request will contain a set desired speed. The module will send an ACK to the requester. If the set speed is less than 25mph, the request is dropped. Otherwise, it is set.

3.2. **Cruise Maintain**: The simple cruise module constantly receives data from the speedometer regarding current speed. The module will ACK this data. If the current speed is greater than the desired speed, the module sends stop requests to the accelerator and activate requests to the brakes. If the current speed is less than the desired speed, the module sends stop requests to the brakes and activate requests to the accelerator. If either the brake pedal or accelerator pedal are manually depressed this function is temporarily disabled.

3.3. **Cruise Suspend**: The simple cruise module will receive data whenever the brake or accelerator pedals are manually depressed. The module will ACK this data. If either of the pedals are depressed, the cruise maintain functionality is temporarily disabled. A button press can also cause this same effect.

3.4. **Cruise Resume**: The simple cruise module will receive data from a button press to resume maintaining the previously saved speed. The module will ACK this request to the button module. The module will then re-activate the cruise maintain functionality.

3.5. **Speed Increase**: The simple cruise module receives data via button press or FDM module to increment the desired speed. An ACK is sent to the requester. The module updates the saved speed.

3.6. **Speed Decrease**: The simple cruise module receives data via button press or FDM module to decrement the desired speed. An ACK is sent to the requester. The module updates the saved speed.

3.7. **Cruise Disable**: The simple cruise module receives data via button press to disable cruising functionality. The module will ACK this data. The saved speed record is cleared, and if activated, the cruise maintain functionality is disabled.
3.8. **Speed Display**: The simple cruise module requests the output module to display the currently desired speed. The output data may originate internally or come from the FDM module request. The module will ACK the request. This occurs momentarily whenever the desired speed is increased or decreased, when the cruise is activated, or when the cruise is resumed.

4. Following Distance Management Requirements
4.1. **FDM Enable**: The FDM module receives a request to activate from the button input module. The request will contain a set desired distance (1-close to 4-very far). The module will send an ACK to the requester and save the desired distance.
4.2. **FDM Calculate**: The FDM will request the current speed from the speedometer and distance from the distance monitor. The FDM will use these variables and desired distance to calculate a minimum-time-to-intercept.
4.3. **FDM Maintain:** The FDM module constantly receives data from the minimum-time-to-intercept calculations. The module will ACK this data. If the current MTTI is greater than desired, the module sends speed increase requests to the simple cruise module. If the current MTTI is less than desired, the module sends speed decrease requests to the simple cruise module.
4.4. **Distance Increase**: The FDM module receives data via button press to increment the desired distance. An ACK is sent to the requester. The module updates the saved distance. If the distance is already maximized (distance = 4), the request is ignored.
4.5. **Distance Decrease**: The FDM module receives data via button press to decrement the desired distance. An ACK is sent to the requester. The module updates the saved distance. If the distance is already minimized (distance = 1), the request is ignored.
4.6. **FDM Disable:** The FDM module receives data via button press to deactivate. The module will ACK this data. The saved distance record is cleared, and if activated, the FDM maintain functionality is disabled. This event does not affect the status of simple cruise maintain.
4.7. **Distance Display**: The FDM module sends data to the simple cruise module regarding desired following distance. The simple cruise module will use this data to display the following distance on the output module.
4.8. **Alert:** If the MTTI is deemed critically low (based on threshold), the FDM will send data to the simple cruise module to output an alert to the operator.
4.9. **FDM Structure Note**: Since the FDM module is built upon simple cruise, and acts as a hypervisor to it, there is no need for the FDM to interact with brake and accelerator pedal depressions. These interact only with simple cruise.

5. Automatic Emergency Brake Requirements

5.1. **AEB Calculate:** The AEB will request the current speed from the speedometer and distance from the distance monitor. The AEB will use these variables and desired distance to calculate a minimum-time-to-intercept.
5.2. **AEB Enable:** If the MTTI is deemed critically low (based on threshold), the FDM will send data to the brakes to apply maximum brakes (desired speed = 0mph).
5.3. **AEB Disable:** Once the MTTI indicates the vehicle is motionless, the FDM will send data to the brakes to stop braking.
5.4. **AEB Independence Note**: The functionality of AEB must be completely independent of the existence of simple cruise and/or FDM. Therefore, some of the requirements of AEB are redundant with those in FDM.

6. Intrusion Prevention System (IPS) Requirements
6.1. **Data Analyze**: The IPS will receive any communications originating from an external source and will ACK the incoming traffic to the sender. The IPS will then forward these communications to the firewall for analysis and processing. If the firewall determines the packet(s) were benign, the IPS will forward the packets to their original intended destination. If the firewall deems the packets malicious, the IPS will drop the packets.

4 Modeling Requirements

There are many levels to the scalable cruise control solution; however only a few of these are openly accessible to the end user. The following use case diagrams illustrate how the user base or large modules interact with the SCC. Note that individual components such as the brake, accelerator, button input, and dashboard output are already implemented and abstracted by the vehicle interface. For this reason, use cases for these components are not within the scope of the project.

Figure 3 displays how a user would interact with various functionalities of the simple cruise module. At surface level, the motorist is able to enable, disable, suspend, and resume the cruise, as well as configure the desired speed. The speed maintain requirement is not seen on the surface but acts as an intermediary between the cruising functions and individual component functions.

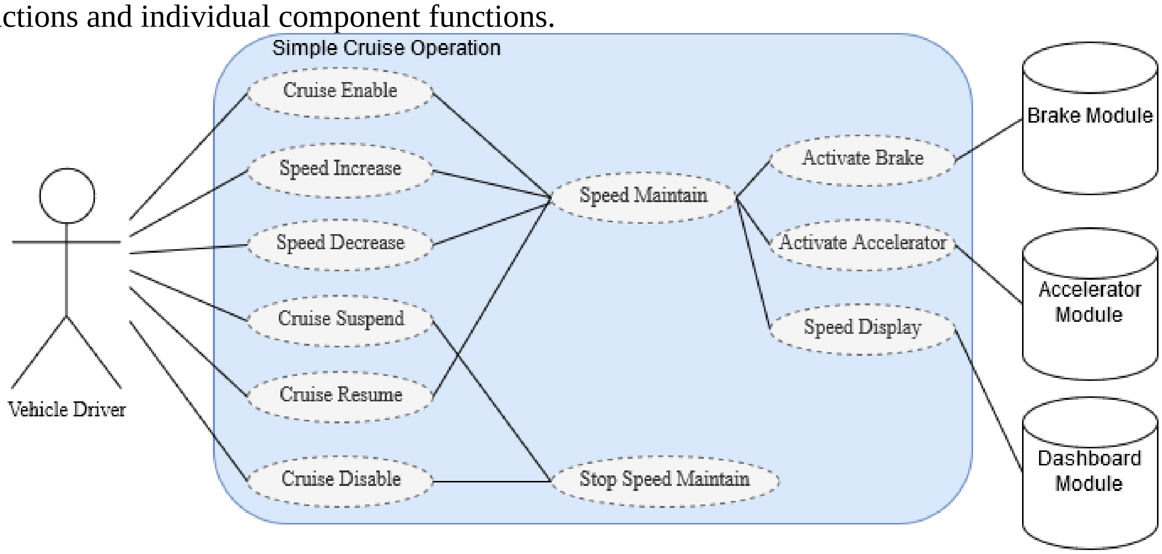

Figure 3: Interaction with Simple Cruise Module

Figure 4 displays the functionality of the auxiliary FDM module. The user is presented with the ability to enable, disable, and modify the desired following distance. Since FDM is built upon simple cruise, the FDM disable function simply returns control to the cruising module in Figure 3. Similar to speed maintain, FDM maintain act as an intermediary. FDM maintain combines the results of FDM calculate and user input to forward instructions to simple cruise. FDM maintain also has control over displaying alerts.

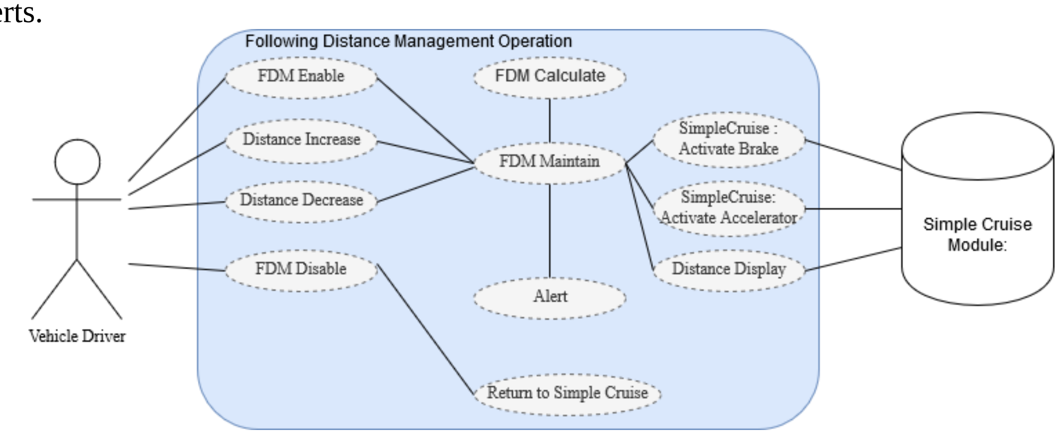

Figure 4: Interaction with Following Distance Management Module

Figure 5 displays how the vehicle's internal computers interact with the automatic emergency brake module. AEB calculate will determine when to enable or disable emergency breaking. The AEB and AEB disable functionalities will then communicate with the brake module and give it appropriate instructions.

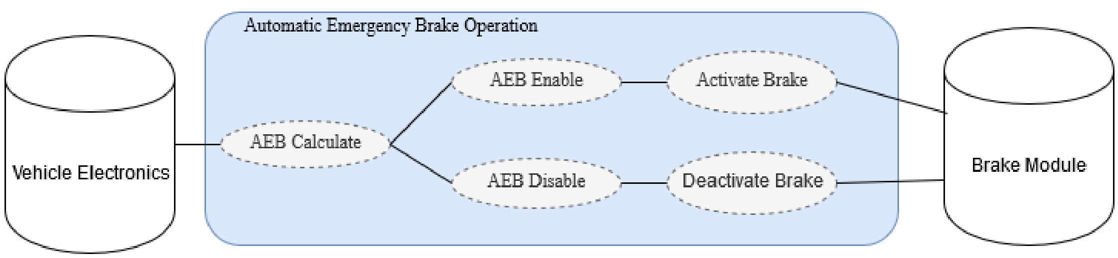

Figure 5: Interaction with Automatic Emergency Brake Module

Figure 6 displays the IPS countermeasures implemented in SCC. The IPS module uses the data analyze functionality to send data to the firewall via traffic receive. The firewall will determine the nature of the data via traffic analyze, and report back with either a traffic allow or traffic deny status.

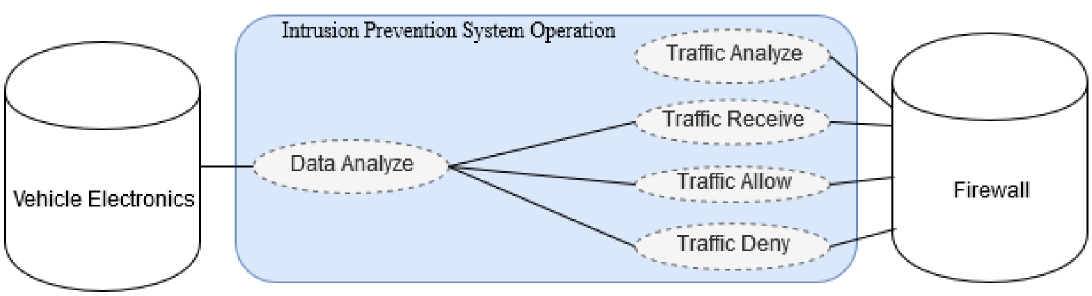

Figure 6: Interaction with Intrusion Prevent System

Figure 7 depicts a suggested overall class diagram for an SCC implementation. Abstract classes are presented as yellow, instantiable classes in blue. Associations between classes are shown as black lines. For clarity purposes, the associations needed by the simple cruise module are shown in red, and associations by the following distance management module are in magenta. Inheritance is illustrated as a hollow white arrow. Composition (ownership) is shown via solid black diamonds. Multiplicity is listed for each association. More detail regarding each class' operations, attributes, and associations is listed in the data dictionary below.

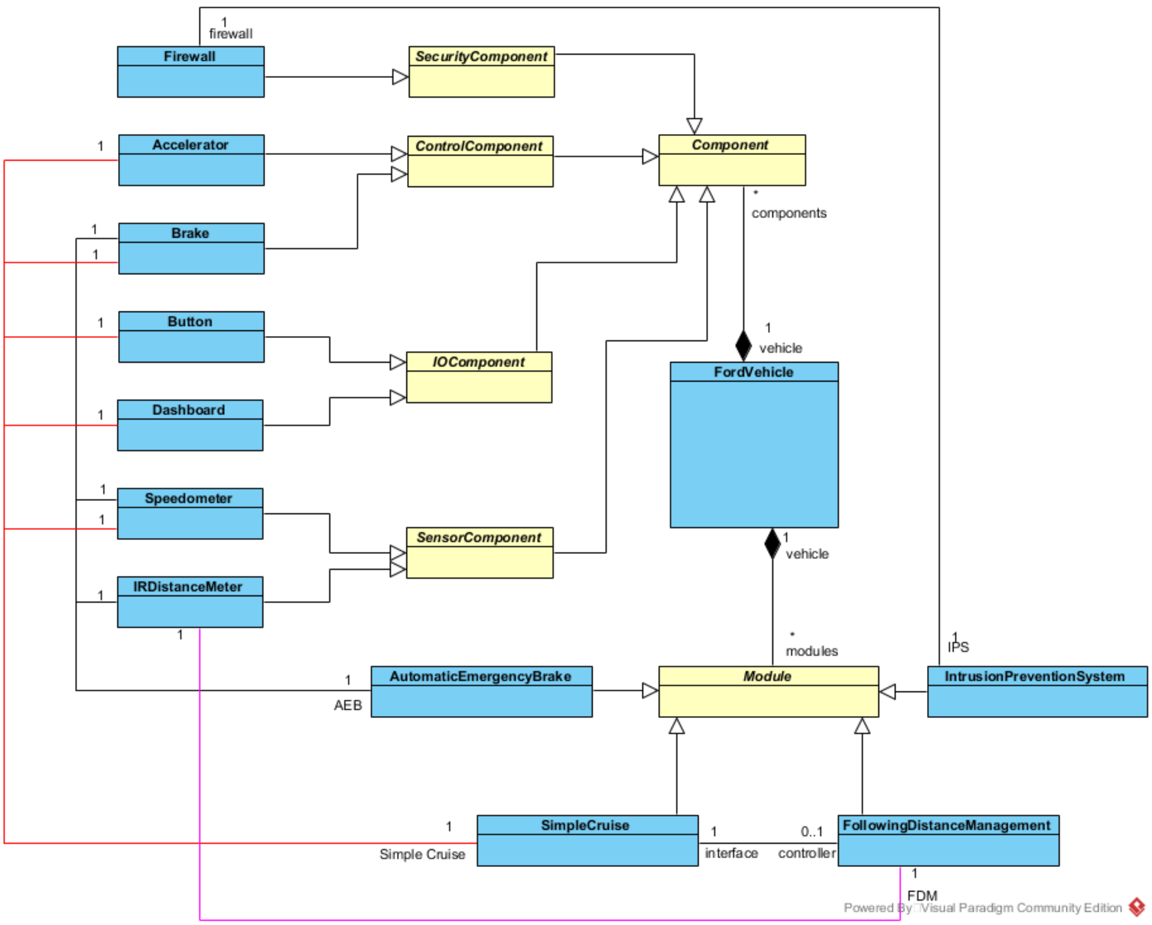

Figure 7: Class Diagram

Element Name		**Description**
Accelerator		This component handles the acceleration of the vehicle. Its used to control increasing speed.
Attributes		
	activated : bool	Controls weather the accelerator is on or off.
	accelerationAmount : int	How much acceleration is necessary.
Operations		
	activate (int): void	Will open the fuel valve and increase the speed of the car. Determines how

		much to increase speed based on desired speed param.
Relationships	This class is inherited from ControlComponent. It is used by the SimpleCruise classes.	

Element Name		**Description**
AutomaticEmergencyBrake		This module handles the AEB aspect of the SCC. Will enable brakes if a collision is imminent.
Attributes		
	MTTI : double	Minimum time to intercept, a value calculated based on speed and distance of when a collision will occur.
	threshold : double	The value, that if MTTI falls below, will call AEBEnable()
Operations		
	AEBCalculate(): void	Calculates the MTTI and compares it to threshold.
	AEBEnable() : void	Is called by AEBCalculate if MTTI < threshold. Calls braking module activate.
	AEBDisable() : void	Called by AEBCalculate once MTTI >= threshold. Calls breaking module to deactivate brakes.
Relationships	This class is inherited from Module. It has associations to IRDistanceMeter, Speedometer, and Brake.	

Element Name		**Description**
Brake		This component handles the braking of the vehicle. It is used to decrease speed when necessary.
Attributes		
	activated: bool	Controls whether the brake is on or off.
	brakeAmount : int	How much brake power is necessary
Operations		
	activate (int): void	Apples pressure to the brake pads and decreases the speed of the car. Determines how much to decrease speed based on desired speed param.
Relationships	This class is inherited from ControlComponent. It has associations to AutomaticEmergencyBrake and SimpleCruise	

Element Name	**Description**

Button		Is the input facet for the various modules of SCC
Attributes		
	latched : bool	Applicable for latching buttons, indicates the button is continuously depressed.
	timeDepressed : int	Applicable for non-latching buttons. The amount of the time the button has been held down
Operations		
	press (): void	Called when button hits sufficient pressure. Will forward call to relevant module. Updates timeDepressed
	latch():void	Called when button is depressed and remains that way.
	unlatch() : void	Called when a depressed button becomes undepressed.
Relationships	This component is inherited from IOComponent. It has associations with SimpleCruise	

Element Name		**Description**
Component		Abstract class for any vehicle component
Attributes		
Operations		
Relationships	The component class inherits the SecurityComponent, ControlComponent, IOComponent, and SensorComponent classes.	

Element Name		**Description**
ControlComponent		Abstract class for any component that controls vehicle driving functions
Attributes		
Operations		
Relationships	The component class inherits the Accelerator and Brake classes	

Element Name		**Description**
Dashboard		Is the output facet for the various modules of SCC.
Attributes		
	content : string	The content to display on the dashboard
	xCoordinate: int	Where, horizontally, to start displaying the content
	yCoordinate: int	Where, vertically, to start displaying the content

Operations		
	display (string): void	Updates the display with data
Relationships	This class is inherited from IOComponent. It has associations with SimpleCruise.	

Element Name		**Description**
Firewall		Controls the network traffic to the vehicle and is used for cybersecurity purposes.
Attributes		
	packet: string	The plaintext contents of a network packet
Operations		
	trafficReceive(string) : void	Receives a packet and stores it in packet
	trafficAnalyze() : void	Analyzes traffic and calls either trafficAllow() or trafficDeny() based on heuristic patterns.
	trafficAllow() : bool	Returns true, indicating that the traffic is not malicious and should be forwarded.
	trafficDeny() : bool	Return false, indicating that the packet is malicious and should be dropped.
Relationships	Is inherited from SecurityComponent. Has associations with IntrusionPreventionSystem.	

Element Name		**Description**
Following DistanceManagement		Controls the following distance of the vehicle when in cruise mode.
Attributes		
	followingDistance : int	The distance to follow the preceding vehicle from short (1) to very long (4)
	MTTI : double	Minimum time to intercept, a value calculated based on speed and distance of when a collision will occur.
	threshold : double	The value, that if MTTI falls below, will call alert()
Operations		
	FDMEnable(int) : void	Enables the following distance management module with an initial following distance (1-4).
	FDMCalculate (): int	Sets MTTI and FDM values based on speed and distance data.

	FDMMaintain(int): void	Receives the MTTI and takes appropriate action. Communicates with SimpleCruise and may send brake or accelerate commands to keep following distance as desired.
	distanceIncrease():void	Increments the following distance by 1 step if possible
	distanceDecrease ():void	Decrements the following distance by 1 step if possible
	alert()	Sends alert output to simpleCruise if the MTTI falls below the threshold
	distanceDisplay()	This function sends display output to simpleCruise when the desired following distance is changed
Relationships	Is inherited from Module. Has associations with SimpleCruise, which it uses to interact with other components, and IRDistanceMonitor, to track following distance.	

Element Name		**Description**
FordVehicle		This is the object that uses all the components and modules.
Attributes		
	year: int	The year the vehicle was manufactured
	model: string	The model of the vehicle
Operations		
Relationships	The vehicle owns (composition) all of the modules and components present in the SCC solution.	

Element Name		**Description**
IntrusionPreventionSystem		This module prevents from outside interference that may alter the components of the vehicle
Attributes		
	traffic : string	A stream of external traffic to enter the vehicle.
Operations		
	dataAnalyze(string) : void	Interacts with the firewall to get its heuristic reports, forward or drops packets, and saves param into traffic attribute.
Relationships	This class is inherited from module. It has associations to Firewall.	

Element Name	**Description**
IOComponent	Abstract class for any component that

		controls input and output
Attributes		
Operations		
Relationships	The component class inherits the Button and Dashboard classes.	

Element Name		**Description**
IRDistanceMeter		This is the infrared distance meter that senses the distance between this vehicle and the next.
Attributes		
	distance: double	This is the value that is returned by the IR sensor
Operations		
	updateData (double): void	This function continually probes the sensor to update the distance attribute.
	getDistance() : double	This function returns the distance attribute
Relationships	This class is inherited from SensorComponent. It has associations with AutomaticEmergencyBrake and FollowingDistanceManagement	

Element Name		**Description**
Module		Abstract class for any of the facets in an SCC implementation.
Attributes		
Operations		
Relationships	The component class inherits the AutomaticEmergencyBrake, SimpleCruise, FollowingDistanceManagement, and IntrusionPreventionSystem classes.	

Element Name		**Description**
SecurityComponent		Abstract class for any component that controls cybersecurity
Attributes		
Operations		
Relationships	The component class inherits the Firewall class.	

Element Name		**Description**
SensorComponent		Abstract class for any component that controls raw data input
Attributes		
Operations		
Relationships	The component class inherits the Speedometer and IRDistanceMeter classes	

Element Name		**Description**
SimpleCruise		This is the actual pre-existing cruise control system used by the vehicle to control the speed of the car
Attributes		
	setSpeed:int	This is the speed that is set for the cruise control
	Threshold : double	
Operations		
	setSpeed(int) : void	Updates the new setSpeed with int param
	speedIncrease() : void	Increments the setSpeed by 1
	speedDecrease() : void	Decrements the setSpeed by 1
	maintainSpeed():void	The function attempts to keep setSpeed as close as possible to currentSpeed by sending requests to the accelerator and brakes. If the difference between the two speeds does not exceed the threshold, this function is not called.
	cruiseEnable	
	DisplayData(varList):void	Sends varList to the display
	cruiseDisable():void	This will deactivate the cruise control and clear saved attributes.
	cruiseEnable(): void	This will activate the cruise control and call setSpee90d to the speedometer reading.
	cruiseSuspend() : void	Temporarily disables cruising functionality but does not clear the saved setSpeed attribute
	cruiseResume() : void	Resumes cruising functionality with the saved setSpeed attribute.
	speedDisplay(string) : void	Receives a string from FollowingDistanceManagement if necessary and forwards it to the dashboard output module.
Relationships	This class is inherited from Module. It has associations with Speedometer, Dashboard, Button, Brake, FolowingDistanceManagement, and Accelerator	

Element Name		**Description**
Speedometer		This is the sensor that tracks the current speed of the vehicle.
Attributes		

	speed: double	Stores the current speed
Operations		
	updateData (double) : void	Continually probes the sensor and updates the speed attribute.
	getSpeed() : double	Returns the speed attribute
Relationships	This class is inherited from IOComponent. It has associations with AutomaticEmergencyBrake and SimpleCruise.	

Figure 8: Data Dictionary

Figure 9 depicts a sequence diagram for a scenario in which a motorist would like to utilize simple cruise to increase the speed at which they are traveling. The motorist has already enabled simple cruise and would like to travel faster. They begin via button press, which increments the desired speed in the simple cruising system. The dashboard informs the user they have changed their desired speed. The speed maintaining function then uses speedometer data and will send an activation signal to the accelerator once the current speed drops below the set speed. The maintaining function will also request the accelerator to stop once the current speed exceeds the set speed. A use case diagram for simple cruise can be found on figure 3.

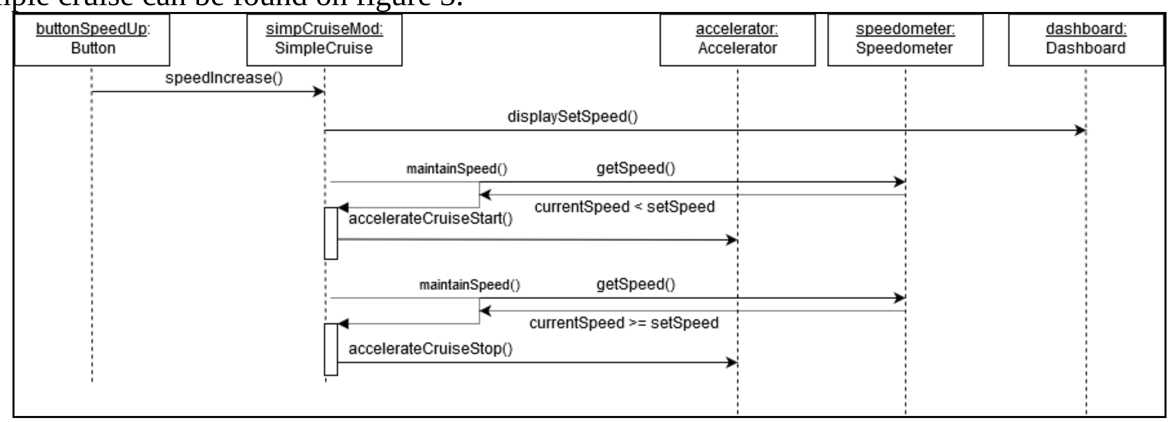

Figure 9: Simple Cruise Sequence Diagram

Figure 10 depicts a sequence diagram in which a motorist would like swap from simple cruise mode to FDM mode. The motorist has already enabled simple cruise but has not yet interacted with any of the FDM functionality. Soon after enabling FDM, the MTTI becomes critically low and an alert is displayed. The process is initialized via button press. This triggers the FDM enable functionality. The FDM module will attempt to maintain distance based on calculations from the distance meter. If the MTTI falls below a set threshold, an alert to displayed to the user. The simple cruise module acts as an intermediary for this output. A use case diagram for following distance management can be found on figure 4.

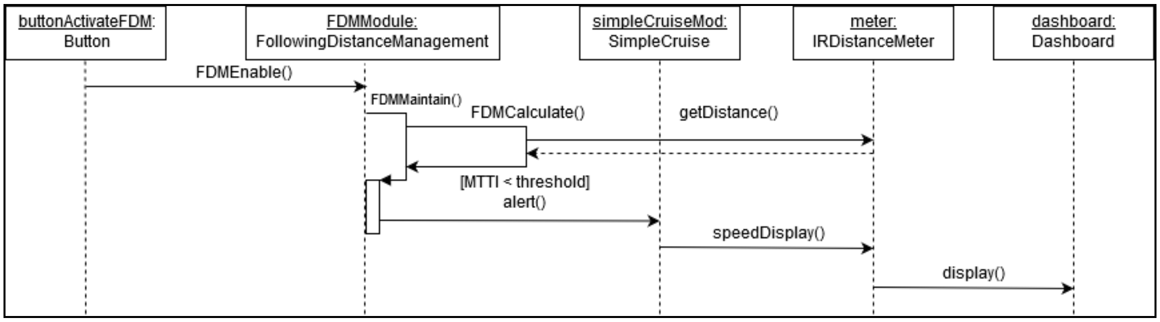

Figure 10: Following Distance Management Sequence Diagram

Figure 11 depicts a sequence diagram in which a vehicle is rapidly approaching a collision and the AEB module activates. The AEB continuously calculates the MTTI based on speed and trailing distance and searches for imminent collisions. When the MTTI falls below the threshold, an activation signal is sent to the braking module. A use case diagram for simple cruise can be found on figure 5.

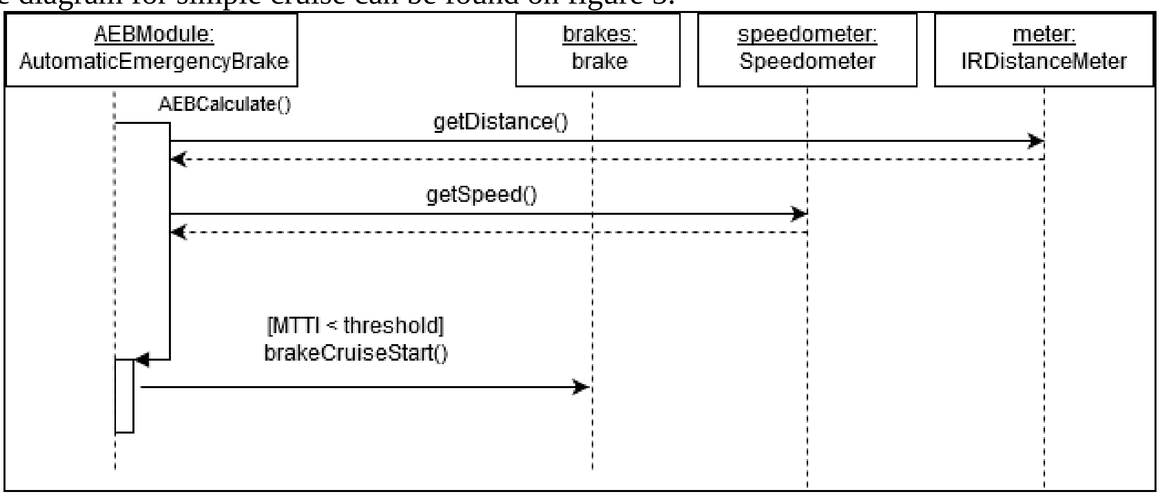

Figure 11: Automatic Emergency Brake Sequence Diagram

Figure 12 depicts a sequence diagram in which the intrusion prevention system of a vehicle receives unknown external traffic and needs to analyze it for origin and signs of malicious intent. When a packet is received, the IPS analyzes it by sending it to the firewall. The firewall will receive the packet, perform its own analysis, and return whether the traffic was deemed malicious or not. A use case diagram for simple cruise can be found on figure 6.

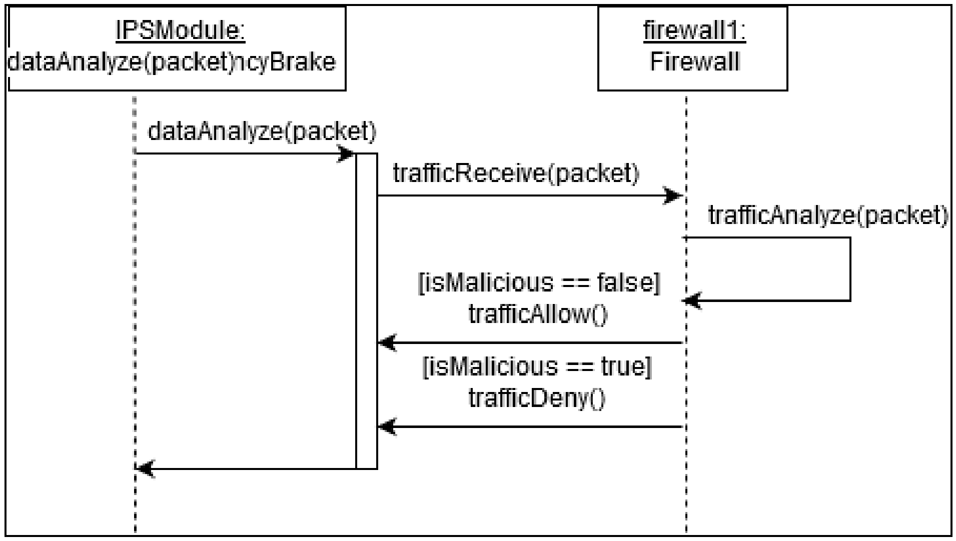

Figure 12: Intrusion Prevention System Sequence Diagram

Figure 13 depicts the changing state of the simple cruising module when undergoing the scenario listed in figure 9. Because the simple cruising module attempts to maintain a constant speed, most state changes are either getter calls or instruction requests to other modules to keep the set speed equal to the current speed.

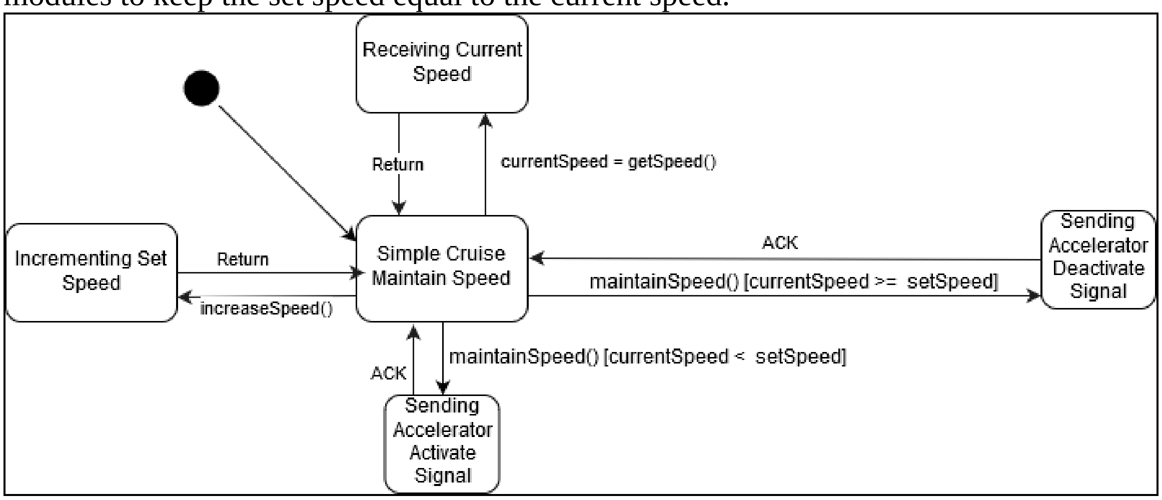

Figure 13: Simple Cruise Scenario State Diagram

Figure 14 illustrates the state of the following distance management module when undergoing the scenario described in figure 10. A button press enables the FDM module, but it is not in action until a following distance has been set. Once the set distance functionality is invoked, the module will attempt to maintain the following distance, similar to speed in simple cruise. The FDM is also constantly calculated the MTTI. If the MTTI falls below the threshold, it will switch to display an alert.

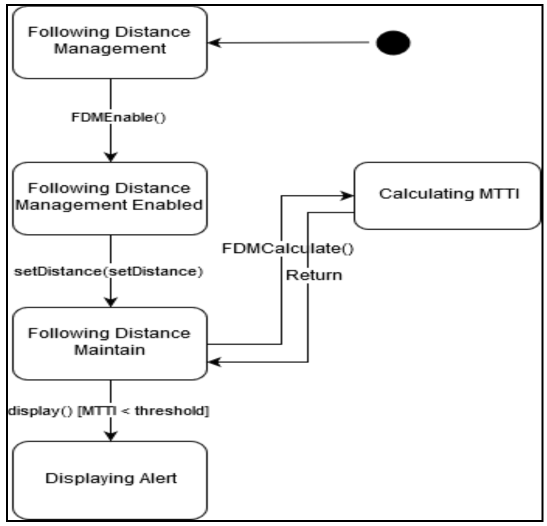

Figure 14: Following Distance Management State Diagram

Figure 15 illustrates the state of the automatic emergency brake module when taking action to prevent a collision. The AEB frequently pings for the distance and speed from their respective components and calculates an MTTI. If the MTTI falls below a set threshold, the AEB will send an activation signal to the braking module.

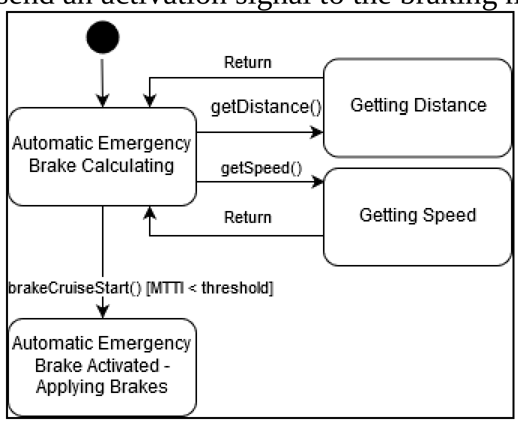

Figure 15: Automatic Emergency Brake State Diagram

Figure 16 shows the state of the intrusion prevention system (IPS) module when handling unrecognized network traffic. The IPS module is dormant when there is not any traffic to handle. Once it is invoked with data analyze, it will forward the packet to the firewall component for inspection. The firewall will do its own inspection, and report back the status of the packet. If it is malicious, the packet is dropped and the IPS returns to idle. If the packet is benign, it is forwarded, and the IPS returns to idle.

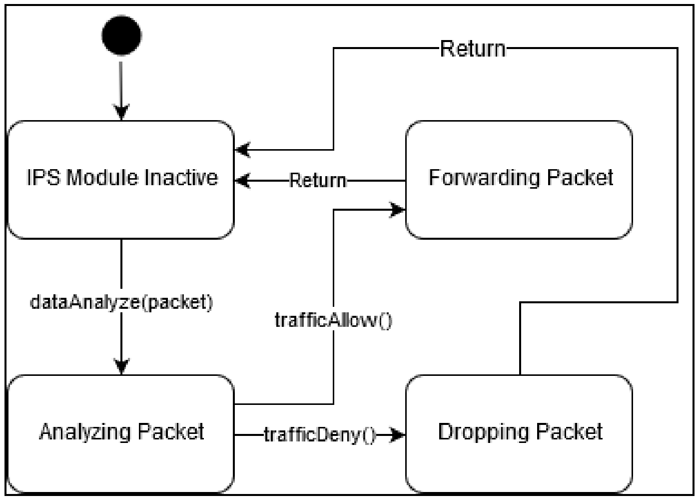

Figure 16: Intrusion Prevention System State Diagram

5 Prototype

A prototype displaying all facets of the SCC implementation is necessary in order to effectively assess risks, elicit feedback, and determine the best design practices. The SCC initial prototype features three main entities.

The first entity depicts a steering wheel (Figure 17, peach-colored wheel). In most vehicles, cruise control functionality is controlled via buttons on the steering wheel. This is no different for SCC implementation. In the prototype, the user is able to depress and latch a number of buttons to interact with SCC. The user is able to enable, disable, increase speed, decrease speed, suspend, and resume a simple cruise emulator. The user is also able to toggle the following distance module and apply similar functions to simple cruise. The steering wheel also features two inputs which serve as a brake and accelerator pedal, respectively.

The second entity depicts the vehicle dashboard (Figure 17, upper grey box). This is where all the user output is displayed. It will show the current speed, and current desired speed or following distance if applicable. If the following distance management and/or automatic emergency brake modules need to display an alert, that is also illustrated on the dashboard.

The third and final entity is the function log (Figure 17, rightmost grey box). This box would not be present in an actual SCC-implemented Ford vehicle, but it useful for demonstration purposes. This box would illustrate the function calls that each module undergoes as the user interacts with the steering wheel and vehicle dashboard. It is intended to help the project developers understand the SCC pipeline.

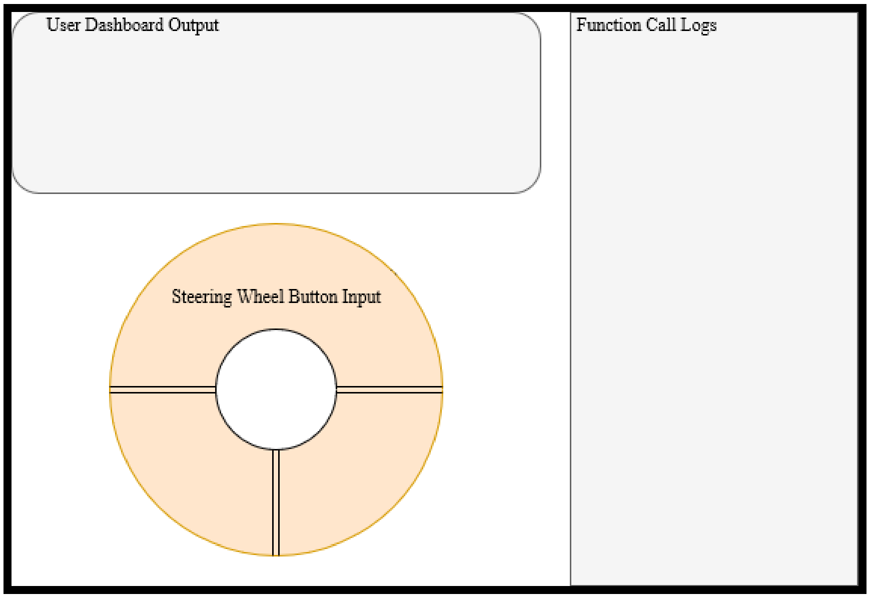

Figure 17: Prototype v1 Interface

5.1 How to Run Prototype

The prototype is intended to be an intuitive and user-friendly interface for those interested in or developing the SCC solution. The prototype is posted publicly on the web, and only involves basic graphics and U/I forms. Therefore, there are minimal software requirements for running the prototype. All the user needs are an up-to-date web browser (Chrome, Firefox, Edge, or Safari recommended) and a working internet connection.

The initial prototype is accessible through the following hyperlink:
http://www.cse.msu.edu/~kuchipu2/#Prototypes

5.2 Sample Scenarios

Suppose a motorist wishes to enable the simple cruise module aspect of the SCC. The first action they would take is to press the button labeled "CRS ON" (Figure 18, green oval). The dashboard would then indicate to the user that simple cruise was enabled, along with their current speed and desired speed. For additional information, the function call logs entity displays the entire pipeline that SCC Follows. This scenario illustrates how SCC probes the current speed, compares it to the desired speed, and sends signals to the accelerator to increase speed.

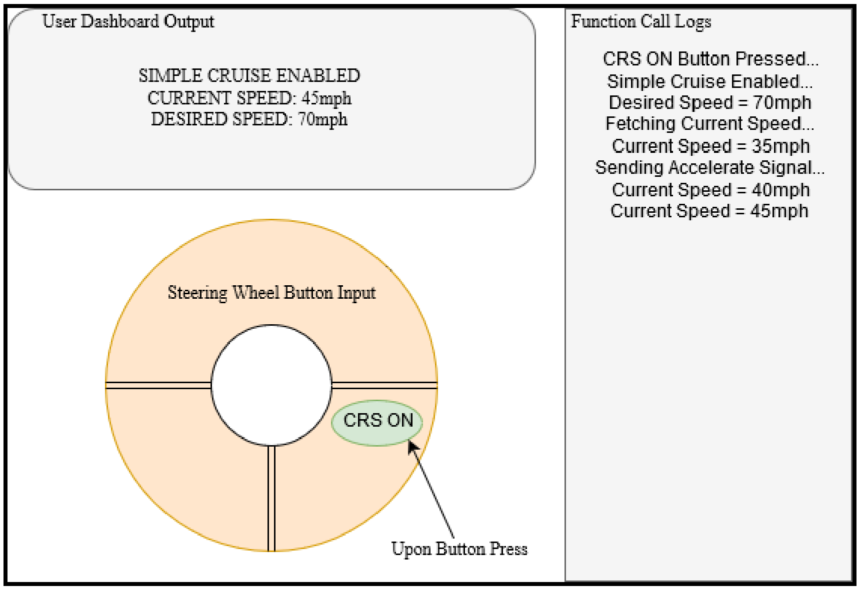

Figure 18: Prototype v1 Cruise Enable Scenario

6 References

[1] *IEEE Recommended Practice for Software Requirements Specifications* (Rep. No. 2). (1998, July 25). Retrieved November 8, 2018, from Software Engineering Standards Committee of the IEEE Computer Society website.

[2] Murray, I., Kuchipudi, P., Chan, K., Brooks, B., & Liang, Z. (2018, October 31). *Scalable Cruise Control Project Requirements Definition* (Rep. No. 3). Retrieved November 8, 2018, from James Daly website.

[3] Winder, E. (2018). *Scalable Cruise Control* (Rep. No. 1). Dearborn, MI: Ford Motor Company website.

[4] Prudhvi, K. (2018, October 31). Scalable Cruise Control Projects. Retrieved October 31, 2018, from http://www.cse.msu.edu/~kuchipu2/

7 Point of Contact

For further information regarding this document and project, please contact **Prof. Betty H.C. Cheng** at Michigan State University (chengb at cse.msu.edu). All materials in this document have been sanitized for proprietary data. The students and the instructor gratefully acknowledge the participation of our industrial collaborators.