Requirements Analysis Document

Sean Albright, Dong Kang, Doug Elkins, Homan Law.

Adaptive Cruise Control (ACC)
ACC Group 4

November 16, 2001
Contents

1 Introduction 3
  1.1 Problem Description ........................................... 3
  1.2 Motivation ..................................................... 4

2 Overview 4

3 Requirements 5
  3.1 Radar Information .............................................. 5
  3.2 Vehicle Information .............................................. 5
  3.3 Adaptive Cruise Control System Information ................... 6

4 UML Analysis 7
  4.1 Use Cases ..................................................... 7
  4.2 Use Cases ..................................................... 7
  4.3 Use Cases ..................................................... 8
  4.4 Use Cases ..................................................... 8
  4.5 Use Cases ..................................................... 9
  4.6 Use Cases ..................................................... 9
  4.7 Object Model .................................................. 10
    4.7.1 Class Overview .......................................... 10
    4.7.2 Data Dictionary .......................................... 11
  4.8 Dynamic Model ............................................... 13
    4.8.1 State Diagrams .......................................... 14
    4.8.2 Sequence Diagrams ..................................... 19

5 Promela Specification for System 20
  5.1 Model Checking and Promela Overview .......................... 20
  5.2 Promela Model ................................................ 20
1 Introduction

At least one person dies in a crash every minute on average. The drivers in these accidents are unable to stop their vehicles in time to prevent a rear-end collision. Air bags and seat belts save ten thousands of people. However, the best solution for preventing these deaths is to keep cars from colliding the first place. Adaptive Cruise Control (ACC) is used to alert the drivers to traffic ahead and avoid collisions when the car is in danger.

This document includes the use case diagram, object model, dynamic model and sequence diagram that can show how the components of our ACC system react with each other.

The Use Case diagram shows the general cases of interaction among the system and the external objects. The Use Case Diagram relies on the event flows between the components. Use Case Diagrams capture a wide view of the primary functionality of the system in a manner easily grasped by nontechnical users. The Use Case diagrams can become a centralized roadmap of the system usage scenarios for people specifying the requirements of the system.

The Object Model describes the structure of the ACC system(objects, attributes, operations, inheritance and aggregations), along with that can be found with the data dictionary which describes the classes amd object model.

The Dynamic Model describes how the objects change state/how the attributes change), in which order the state changes can take place and how events are handled.

The Sequence Diagrams are used to capture and expand on use-cases and demonstrate communicating concurrency. It can highlight the sequencing of the interactions between objects.

All these diagrams can give you a clear idea about the Adaptive Cruise Control in a vehicle.

1.1 Problem Description

The main idea behind collision avoidance is to measure the clear distance ahead of a car with radar, and apply brakes if the car will come within an unsafe distance of the lead vehicle. Typically, these systems are linked with the cruise control and automatic speed control systems in the car to hold the car at or below the minimum of the leading vehicle speed and the speed being desired by the driver or cruise control.

There are two main cases: In the first, the car is operating in the clear with the cruise control set. If a vehicle moves in front of the car at a distance and speed considered unsafe, the car needs to decelerate with brakes until it is at least at the safe distance. Typically, the safe distance is measured in seconds of separation, for example, a two second space at 60 M.P.H. would represent 176 feet of separation. The car may also close on a slower moving lead vehicle. In this case, the car must decelerate before arriving at the unsafe distance.

In the second case, the cruise is not set and speed is controlled by the driver. In this case, the car must also not come closer than the unsafe distance from the lead vehicle by overriding the driver through use of brakes, reduction in speed commanded to the speed control, or both. When
there is no lead vehicle, the car can assume whatever speed the driver is demanding.

1.2 Motivation

In 1999, more than 6 million crashes occurred on U.S. highways, killing over 41,000 people and injuring nearly 3.4 million others. Rear-end collisions accounted for almost one-third of these crashes (1.848 million) and 11.8 percent of multivehicle fatal crashes (1,923). Regardless of the individual circumstances, the drivers in these accidents were unable to detect slowed or stopped traffic and to stop their vehicles in time to prevent a rear-end collision. Hence, the ultimate solution, and the only one that save far more lives, limbs, and money, is to keep cars from smashing into each other in the first place. The technology to alert drivers to traffic ahead and protectively avoid collision is the Adaptive Cruise Control (ACC).

Adaptive Cruise Control (ACC), like the traditional cruise control, maintains the driver’s speed. However, it will also adapt the vehicle speed to follow a slower vehicle ahead, and resume when the vehicle is no longer ahead. ACC informs the driver about its operation and will apply limited braking when required.

In this document, we are going to tell how the radar-based ACC system works according to our simulation. We use radar-based system instead of laser-based system due to the poor performance of laser in rain and snow conditions. Also, this document can tell how the ACC system interacts with the driver, radar, cruise control system, engine control module and the braking system of a vehicle.

2 Overview

Adaptive cruise control system uses traditional cruise control switches and enhanced displays showing the vehicle ahead. The system is similar to the usual cruise control in that it maintains the vehicle’s pre-set speed. However, the benefit is that it automatically adjusts the speed to maintain a proper headway between vehicles in the same lane. This is achieved through a radar headway sensor, digital signal processor, and longitudinal controller.

This ACC system uses the vehicle speed set by the driver as an upper limit for controlling the host vehicle’s speed automatically so it can maintain a certain desired distance from a vehicle ahead.

The ACC system measures the distance to a lead vehicle and the relative velocity of the vehicles, based on information obtained by a radar unit installed at the front of the host vehicle. It uses the data and automatically controls the host vehicle’s speed by activating the throttle actuator or the brake actuators in order to maintain the set distance between the two vehicles. The radar unit transmits a wave pulse and computes the distance to a forward object from the time it takes for the reflected wave to be received. The relative speed is calculated from the difference in frequency between the transmitted and reflected waves.

In the event a preceding vehicle decelerates or another vehicle cuts in front of the host vehicle so that the headway distance is shorter than the value set by the driver, the ACC system automatically closes the throttle valve to decelerate the host vehicle until it returns to the preset distance. When the situation necessitates even greater deceleration, the system also automatically applies
the brakes. Once the headway distance becomes longer than the set distance, as a result of the preceding vehicle or the host vehicle changing lanes, for example, the ACC system automatically opens the throttle valve and accelerates the host vehicle gradually until the set distance is reached. Then, it acts again to maintain the desired headway distance to preceding traffic.

The driver can override the ACC operation by braking or accelerating the vehicle manually. Braking overrides the system at any time if the braking force from the driver is greater than that of the system. In this case, the control system is released and precedence is given to the driver’s action. When the lead vehicle changes lanes or exits and the road is clear, the ACC will accelerate to the set speed.

3 Requirements

3.1 Radar Information

The radar serves as a peripheral source of information for the ACC system. Primarily, it obtains information pertaining to the lead car in relation to the host vehicle. The information obtained is limited to the range of the radar (400 ft.), and consists of the following:

3.1.1 A ”lead vehicle detected” signal

This signal is sent when a lead car is detected within 400 feet of the host vehicle. If there isn’t a car in the 400 feet detection range, this signal is absent.

3.1.2 The distance to the lead vehicle

This data item is transmitted as a scaled integer and measured in feet. It is scaled as a multiple of 100, which allows for two decimal places of precision. For example any data in the format, xx.xx, would be converted, xx.xx * 100 = xxxx, and then transmitted.

A safe distance between the lead vehicle and the host vehicle is considered to be a distance that allows for a two second cushion. This distance can be determined by subtracting the relative speed from the host vehicle’s current speed and multiplying the result by 2 seconds.

3.1.3 The relative or closing speed between the host vehicle and the lead vehicle

This data item is scaled in the same manner as the distance to the lead vehicle (see 3.1.2), but is measured in \( \frac{ft}{sec} \).

3.2 Vehicle Information

3.2.1 Brakes

The brakes are considered to be the host vehicle’s interface for deceleration by the vehicle’s operator and the ACC.

3.2.1.1 When used by the ACC, the brakes have a deceleration capability that ranges from 0 to 12 \( \frac{ft}{sec^2} \). This range is scaled at increments of \( \frac{ft}{sec^2} \).

3.2.2 Engine Control Module (ECM)

The Engine Control Module is the system that monitors and controls the host vehicle’s engine. The following properties describe the pertinent ECM behavior.

3.2.2.1 The ECM will accept a digital speed specification from the Cruise Control Module. The ECM will then maintain that speed until a release signal is received, at which point throttle control will be transferred to either manual, or the ACC system.
3.2.2.2 The speed control on the ECM can be overridden if the speed desired by the vehicle operator via the accelerator is greater than the speed indicated by the speed control.

3.2.3 Cruise Control Module (CCM)

The Cruise Control Module allows the user to indicate to the engine control module to maintain the current speed. The following properties describe the pertinent CCM behavior.

3.2.3.1 When the vehicle operator engages the cruise control, the CCM sends a digital message to the engine control module to set and hold the current speed.

3.2.3.2 The vehicle operator is notified by signal when the cruise control is engaged.

3.3 Adaptive Cruise Control System Information

The adaptive cruise control (ACC) system is the primary focus of this document, and its description can be read above in the Overview in Section 2.

3.3.1 Engagement Conditions

The ACC will only engage or turn on when the following conditions exist:

3.3.1.1 The host vehicle is travelling at a speed greater than 25 miles per hour.

3.3.1.2 A lead car is detected within the 400 feet specified in the Radar Information above.

3.3.2 Engine Control and Brake Override Description

The ACC may override other controls within the host vehicle if conditions exist where it would be beneficial to do so. The following properties describe this behavior.

3.3.2.1 The vehicle operator will be notified by signal when the ACC is overriding vehicle operator controls.

3.3.2.2 When the ACC no longer needs to override vehicle controls, control will return to the appropriate vehicle operator or system.
4 UML Analysis

4.1 Use Cases

4.2 Use Cases

Use case: Set Cruise Control

Actors: Engine Control Unit.

Type: Primary

Description: The Engine control unit remember cruise speed when a driver sets cruise on. Then it keeps the cruise speed as long as it is at the safe distance.

Cross-reference: 3.2.2.1 and 3.3.2.2

Uses:

Extended by:

Extends:
### 4.3 Use Cases

<table>
<thead>
<tr>
<th>Use case:</th>
<th>Listen for Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actors:</strong></td>
<td>Radar.</td>
</tr>
<tr>
<td><strong>Type:</strong></td>
<td>Primary</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>ACC gets a signal when a lead car presents within 400 feet in front of the car.</td>
</tr>
<tr>
<td><strong>Cross-reference:</strong></td>
<td>3.1.1 and 3.3.1.2</td>
</tr>
<tr>
<td><strong>Uses:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Extended by:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Extends:</strong></td>
<td></td>
</tr>
</tbody>
</table>

### 4.4 Use Cases

<table>
<thead>
<tr>
<th>Use case:</th>
<th>Get Current Info</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actors:</strong></td>
<td>Engine Control Unit, Radar.</td>
</tr>
<tr>
<td><strong>Type:</strong></td>
<td>Essential</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>In order to compute brake rate, ACC asks current speed of the host car from engine control unit and the closing speed of the lead car and distance between them when it gets a signal from a radar listener.</td>
</tr>
<tr>
<td><strong>Cross-reference:</strong></td>
<td>3.1.2 and 3.1.3</td>
</tr>
<tr>
<td><strong>Uses:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Extended by:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Extends:</strong></td>
<td></td>
</tr>
</tbody>
</table>
4.5 Use Cases

<table>
<thead>
<tr>
<th>Use case: Compute Brake Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actors:</strong> Radar, Engine Control Unit, Brake.</td>
</tr>
<tr>
<td><strong>Type:</strong> Essential</td>
</tr>
<tr>
<td><strong>Description:</strong> ACC computes a brake rate with current info from engine control unit and radar unit when the speed of vehicle is over 25 M.P.H. ACC displays its activity on a driver's display when it acts.</td>
</tr>
<tr>
<td><strong>Cross-reference:</strong> 3.2.1.1 and 3.3.1.1</td>
</tr>
<tr>
<td><strong>Uses:</strong></td>
</tr>
<tr>
<td><strong>Extended by:</strong> Set Brake Rate</td>
</tr>
<tr>
<td><strong>Extends:</strong> Get Current Info</td>
</tr>
</tbody>
</table>

4.6 Use Cases

<table>
<thead>
<tr>
<th>Use case: Set Brake Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actors:</strong> Radar, Engine Control Unit, Brake</td>
</tr>
<tr>
<td><strong>Type:</strong> Primary</td>
</tr>
<tr>
<td><strong>Description:</strong> ACC takes over control of the vehicle.</td>
</tr>
<tr>
<td><strong>Cross-reference:</strong> 3.2.1.1 and 3.3.2.1</td>
</tr>
<tr>
<td><strong>Uses:</strong></td>
</tr>
<tr>
<td><strong>Extended by:</strong></td>
</tr>
<tr>
<td><strong>Extends:</strong> Compute Brake Rate</td>
</tr>
</tbody>
</table>
4.7 Object Model

Figure 2: ACC Object Diagram

4.7.1 Class Overview

Figure 2 shows the Objects in use in the system, their relationships and interactions. While this document is mainly concerned with the ACC unit itself, it is valuable to understand its interactions with other vehicle components so they have been included as well. This section will briefly describe those relationships and interactions.

- There are essentially 5 components shown in Figure 2, the ACC, the Radar unit, the UI, the Brakes, and the Engine Control Unit.

- The ACC calls the setBrake() method of the Brakes to decelerate the host vehicle, using information it has received from the Radar unit and the Engine Control Module.

- The Brakes, in turn, update the current speed in the Engine Control Unit to decelerate at the rate provided by the ACC.

- The UI provides the driver the ability to accelerate and decelerate the car by interacting with the ECM and Brakes to change the speed. It also displays information provided by the ACC, such as imminent collision warnings.

- Two of these main units are composed of multiple objects, the ACC and the Engine Control Module.
• The ACC contains a separate object called the Radar Listener. This Listener controls communication between the ACC and the Radar unit by listening for interrupts from the Radar unit.

• The Engine Control Module consists of two other objects, the Cruise Control and the Manual Throttle Control. The Cruise Control handles Cruise functions for the engine and interacts with the UI to set the speed and the ACC to be reset after the ACC has been engaged. The Manual Throttle Control interacts with the UI to allow the driver manual control of the acceleration of the Host Car when the Cruise Control unit is not engaged.

4.7.2 Data Dictionary

This section defines all the objects, methods, and attributes that are found in the Object Diagram 2 and is subdivided by object.

ACC - (Adaptive Cruise Control) An automatic collision avoidance system which reads data from the Radar and the Engine Control Module to determine when and at what rate braking is necessary to avoid collision with a leading vehicle. The ACC will maintain at least 2 seconds of separation with the leading car, but will not initialize unless the host car is travelling faster than \(25 \frac{ft}{sec}\). When the ACC is active and decelerating, the ACC alerts the driver and when the car is no longer a threat, returns the host car either to its Cruise Control or Manual Control state from prior to braking.

• calcSafeRate - This method determines the appropriate rate of deceleration to match the speed of the host vehicle to that of the leading vehicle and position it a safe distance (2 seconds of separation) away. Its calculations are based on the distance and relative speed of the leading vehicle from radarInfo and it is limited to integral rates of deceleration from \(0 \frac{ft}{sec^2}\) to \(12 \frac{ft}{sec^2}\). If the host car is in the danger zone it recommends the full deceleration of \(12 \frac{ft}{sec^2}\), otherwise it uses the formula:

\[
rate = \frac{\frac{1}{2} \text{Relative Speed}^2}{\text{Distance to Lead Car} - (2 \text{sec of separation})(\text{Leading Car Speed})}
\]

to determine a decimal deceleration rate, which it then rounds up to the nearest integer.

• Cruise - This integer variable holds the Cruise Control speed of the car prior to the activation of the ACC so that the host car can resume its original Cruising Speed when the ACC returns control. This value is provided by calling the Cruise Control method getCruiiseSpeed().

• currentSpeed - This integer variable holds the current speed of the host vehicle as provided by the getSpeed() method of the Engine Control Module for use in deceleration calculations.

• radarInfo - An array of two integers, the first representing the current distance to the leading vehicle, the second its relative speed. These data are provided by the method updateInfo() of the Radar unit.

• WasCruiseOn - This boolean variable holds the value returned from the Cruise Control by getCruiise() to determine if the ACC should resume a Cruising Speed or manual acceleration when it returns control of the host car.
**Brakes** - These take in deceleration rates from the driver of the host vehicle and from the ACC and decrease the speed of the host car accordingly. The Brakes constantly decelerate the vehicle at the BrakeRate, which is 0 when neither the driver nor the ACC is decelerating.

- **BrakeRate** - This integer variable holds the current rate at which the Brakes should slow the car.
- **setBrake(rate)** - This method sets the BrakeRate variable to equal rate.

**Cruise Control** - This is a component of the Engine Control Module responsible for maintaining the speed of the host car automatically when it is engaged.

- **CruiseOn** - This boolean is true when the driver has engaged the Cruise Control and false when the Cruise Control is turned off.
- **CruiseSpeed** - This is the Speed the driver of the host car wishes the Cruise Control to maintain.
- **getCruise()** - This method returns the value of CruiseOn, to determine if the Cruise Control is on or not.
- **getCruiseSpeed()** - This method returns the value of CruiseSpeed.
- **setCruise(boolean)** - This method is used to turn the Cruise Control on and off by setting the value of CruiseOn.
- **setCruiseSpeed(speed)** - This method is used to set the CruiseSpeed.

**Engine Control Module** - The ECM is responsible for accelerating and maintaining Engine Speed, as well as managing the Cruise Control and providing information about the current speed of the host vehicle to other units.

- **accelerate(speed)** - This is the function the Engine Control Module employs to gradually bring the engine up to speed.
- **currentSpeed** - This integer variable holds the current speed of the engine.
- **getSpeed()** - This function returns the value of currentSpeed to objects wishing to know the current speed of the host car.

**Manual Throttle Control** - This object is engaged when the Cruise Control is off and control of the acceleration of the engine of the host vehicle is turned over to the driver.

**Radar** - The Radar unit is a sensor mounted on the front of the host vehicle which can detect the presence of an object in its path within 400ft. It also determines the speed of that object relative to the speed of the host vehicle. When the Radar unit acquires a target (which is assumed to be a leading vehicle) it interrupts the Radar Listener to notify the ACC. It sends a similar signal to the Radar Listener when the target has been lost.

- **currentInfo** - This is an array of two integers, the first of which denotes the relative speed of a detected object, the second the distance to the object.
• `getInfo()` - This function returns the value of `currentInfo`.

• `LeadCarDetected` - This is a boolean variable which is true when the Radar detects an object within 400ft. and false otherwise.

• `updateData()` - This function is called repeatedly to update the values in `currentInfo` with realworld data.

**Radar Listener** - This is a component of the ACC which listens for interrupt signals from the Radar unit to signify the acquisition and loss of targets.

• `detected` - This boolean variable is true when the Radar unit has notified the Listener that it has acquired a target and false when the Listener is notified otherwise.

• `listen()` - The main function of the listener, this waits for an interrupt, then updates the value of `detected` and goes back to waiting for an interrupt.

**UI** - This is an implementation specific User Interface which provides controls and notifications to the driver. Some important notifications are the activities of the ACC, crash warnings, and the current state of the Cruise Control.

### 4.8 Dynamic Model

Figure 3 shows the concurrent State Diagrams for the component objects of the Host Car. While this document mainly concerns itself with the ACC, it is again illustrative to include the related components of the vehicle. These State Diagrams show the response of each object to events and conditions that may occur during their lifetime, and are discussed individually below.
4.8.1 State Diagrams

**ACC** Figure 4 depicts the workings of the ACC unit in a state diagram.

- The ACC begins in an Idle state, doing nothing until the Radar Listener reports a leading vehicle has been detected. If the host car is travelling at over $25\frac{ft}{sec}$ then the ACC will collect the information about the Cruise Control and then move into the Engaged composite state.

- The Engaged state is a composite of several states.
  - The starting state is the Running state. Because of its null transition to the Updating state, this state is essentially redundant, but serves to clarify the document.
  - The ACC then moves on to the Updating state, where it gets the current information from the Radar and Engine Control Modules. The ACC determines if it is Gaining or Not Gaining on the Leading Car.
  - If the ACC is Gaining, it determines if it is currently in the Safe Zone or in the Danger Zone. If it is in the Danger Zone, it tells the brakes to decelerate at $12\frac{ft}{sec^2}$ and returns to the Running state to begin the process again. If it is not in the Danger Zone, it calculates the best deceleration rate using the function `calcSafeRate()` and tells the brakes to decelerate at that rate.
  - If the ACC is Not Gaining on the Leading Vehicle, it determines if it is in the Safe Zone or the Danger Zone. If it is in the Danger Zone, it will decelerate at its maximum rate of $12\frac{ft}{sec^2}$ and return to Running. If the ACC determines it is in the Safe Zone, it will merely return to running to begin reassessing the data.
  - If detected becomes false at any time in this Composite state, the ACC transitions out of Engaged and into Exiting.

- From the Exiting state, the ACC checks the information it got from the Cruise Control to determine how to reset the car. If the Cruise Control was previously engaged, then the ACC resets it and returns to Idle, otherwise, it merely returns to Idle.
Brakes Figure 5 depicts the workings of the Brakes unit in a state diagram.

- The Brakes have only one state, the Idle state.
- On entering the Idle state, the Brakes update the Speed of the Engine via the Engine Control Module based on the current value of Brake Rate.
- Two transitions can occur:
  - A call to setBrake(rate) can occur. If this happens, the Brakes set Brake Rate = rate, and then return to Idle.
  - A timeout can occur. After some implementation-determined time, the Brakes transition back to the Idle state, where they update the speed again.
**Engine Control Module**  Figure 6 depicts the workings of the Engine Control Module in a state diagram.

- The Engine Control Module begins in state State. Here it can process requests to get and set the value of Cruise or to get the value the current Speed.
- When setSpeed(speed) function is called, the ECM transitions in the Accelerating state. It cycles here, gradually increasing the Current Speed until it has reached the value of (speed) and then returns to State.
Radar Listener Figure 7 depicts the workings of the Radar Listener in a state diagram.

- The Radar Listener begins in the state Waiting to Acquire, where it waits for the Radar unit to acquire a target and interrupt it. When it is interrupted, it updates the value of detected and moves to the state Waiting to Lose.
- In Waiting to Lose, the Radar Listener waits for the Radar unit to lose its target and send an interrupt. When it receives this interrupt, it returns to the Waiting to Acquire state.
Radar Listener State Diagram

Waiting to Aquire

interrupt()||detected = true

interrupt()||detected = false

Waiting to Lose

Figure 7: Radar Listener State Model

Radar Listener State Diagram

when(Lead Car Detected)||Interrupt(RadarListener

timeout(.001s)||update Data():

getInfo()||return(currentInfo);

Figure 8: Radar State Model

1 Radar Figure 8 depicts the workings of the Radar unit in a state diagram.

2 The Radar unit begins in the Idle state, its only state.

3 Every .001 seconds (or some other value, depending on implementation) it updates its values based on the real world information it gets from the car ahead of it.

4 When a leading car is detected or lost, the Radar unit interrupts the Radar Listener to inform the ACC and returns to Idle.

5 When a call to getInfo() is made, the Radar unit returns the current relative speed and distance of the leading vehicle and then goes back to the Idle state.
4.8.2 Sequence Diagrams

**Scenario 1** Host car is travelling at a speed greater than 25 mph, cruise control is engaged. A lead car pulls out into radar range in "safe zone." Eventually, lead car leaves radar range.

**Figure 9: ACC Sequence Diagram - Scenario 1**

- The radar unit detects a lead vehicle it interrupts the sleeping radar listener. The radar listener then initializes the ACC and goes back to sleep.

- Once activated, the ACC gets the Cruise Control information and begins looping. First it polls the Radar unit and the ECM for information, using the `getInfo()` function of Radar, and the `getSpeed()` function of the ECM.

- Based on the new information from the Radar unit and the ECM, the ACC initializes the appropriate warnings in the GUI and calculates the appropriate deceleration rate.

- The ACC sets the deceleration rate and then reloops.

- When the Radar unit loses the lead car, it interrupts the Radar Listener again which tells the ACC to begin a cleanup sequence.

- The ACC stops braking, resets the cruise control to its original speed, and deactivates all warnings in the GUI. The ACC then goes back to sleep.

**Scenario 2** Host car is travelling at a speed greater than 25 mph. Speed is adjusted manually. A lead car pulls out into radar range in "safe zone." Eventually lead car leaves radar range.
Scenario 3  Host car is travelling at a speed greater than 25 mph, cruise control is engaged. A lead car pulls out into radar range in "danger zone."

5 Promela Specification and System

5.1 Model Checking and Promela Overview

Promela, or the PROcess MEta LAnguage, is a non-deterministic language used to model systems to facilitate validation. State diagrams are coded into the language and can then be run on a model checker such as SPIN. This model checker simulates the system to allow easy observation of its behavior, either interactively or randomly and provides tools to ease the detection and diagnosis of errors in the system. These tools include the detection of unreachable code; assertion testing, which allows logical checks of properties of the model at various points in its lifecycle; and event histories to recreate error conditions.

5.2 Promela Model

We have encoded into Promela the two components of the Dynamic model above in Figure 3 which comprise our part of the system - namely, the ACC itself. These are the main ACC unit and the Radar Listener. The State diagrams to which the above code can be found in Figures 4 and 7 respectively. In general, state machines from dynamic models are coded into Promela as "prototypes". You'll notice that in our code, we have abstracted the composite state Engaged to its own prototype to represent Figure 4 more accurately.

/* states */
mttype {IDLE, WAITING_TO_ACQUIRE, WAITING_TO_LOSE,
    EXITING, RUNNING, UPDATING, NOT_GAINING, BRAKING, GAINING, DANGER_ZONE, SAFE_ZONE};

/* interrupts */
mttype {minterrupt};

/* globals */
mttype RLState; /* This holds the current Radar Listener state. */
mttype ACCState; /* This holds the current ACC state */

bit Detected = 0;
bit InDangerZone =0;
bite CruiseWasOn = 0;
bite SpeedFaster25 = 0;
bite PosClosingSpeed = 0;

chan composite = [0] of {int}; /* Used to communicate between the ACC and its composite state, Engaged */
chan interrupt = [0] of {mttype}; /* Used to send interrupts to the Radar Listener as the Radar unit would. */
/* state.promela */

The init statement sets up the environment and sends signals to the Radar Listener and the ACC. It can be modified to fit the needs of testing */

init
{
    interrupt!minterrupt;
    printf("Car Detected\n");
    SpeedFaster25 = 1;
    printf("SpeedFaster than 25 now\n");
    interrupt!minterrupt;
    printf("Car Lost\n");
}

/* The procotype represents the Radar Listener. It is a fairly straightforward two state model. */

active procotype Radar_Listener()
{
    RLState = WAITING_TO_ACQUIRE;
    do
        :: (RLState == WAITING_TO_ACQUIRE) ->
            printf("In State: WAITING_TO_ACQUIRE\n");
            interrupt?minterrupt -> Detected = 1;
            RLState = WAITING_TOLOSE;
        :: (RLState == WAITING_TOLOSE) -> ;
            printf("In State: WAITING_TOLOSE\n");
            interrupt?minterrupt ->
                Detected = 0;
            RLState = WAITING_TO_ACQUIRE;
    od;
}

/* Here is the bulk of the code, the main ACC unit. */
active procotype ACC()
{
    int pid; /* This is used in communicating between the ACC procotype and its composite state, Engaged. */
ACCState = IDLE;

do
  :: (ACCState == IDLE) ->
    printf("In State: IDLE\n");
    if
      :: Detected -> SpeedFaster25 ->
        printf("GetCruise and GetCruiseSpeed");
        pid = run Engaged(); /* Enter the Engaged state */
        composite??eval(pid); /* Wait until the Engaged state finishes */
        ACCState = EXITING; /* Move on to the exiting state */
    fi;
  :: (ACCState == EXITING) ->
    printf("In State: EXITING\n");
    if
      :: CruiseWasOn ->
        printf("ResetCruise");
        ACCState = IDLE;
      :: !CruiseWasOn ->
        ACCState = IDLE;
    fi;
  od;

/* The proctype Engaged is an abstraction for the
composite state of the same name in the ACC. */
proctype Engaged()
{
  ACCState = RUNNING;

do
  :: (ACCState == RUNNING) ->
    printf("In State: RUNNING\n");
    if
      :: !Detected ->
        break;
      :: Detected ->
        ACCState = UPDATING;
    fi;
  :: (ACCState == UPDATING) ->
    printf("In State: UPDATING\n");
    printf("Get Radar and ECM info");
    if
      :: !Detected -> break;
    fi;

22
if
  :: PosClosingSpeed ->
  ACCState = GAINING;
  :: !PosClosingSpeed ->
  ACCState = NOT_GAINING;
fi;
:: (ACCState == NOT_GAINING) ->
printf("In State: NOT_GAINING\n");
  if
    :: !Detected ->
      break;
fi;

if
  :: InDangerZone ->
  printf("SetBrake(12)"('{}');
  ACCState = BRAKING;
:: !InDangerZone ->
  ACCState = RUNNING;
fi;
:: (ACCState == BRAKING) ->
printf("In State: BRAKING\n");
  if
    :: !Detected ->
      break;
fi;
ACCState = RUNNING;
:: (ACCState == GAINING) ->
printf("In State: GAINING\n");
  if
    :: !Detected ->
      break;
fi;

if
  :: InDangerZone ->
  ACCState = DANGER_ZONE;
:: !InDangerZone ->
  ACCState = SAFE_ZONE;
fi;
:: (ACCState == DANGER_ZONE) ->
printf("In State: DANGER_ZONE\n");
  if
    :: !Detected ->
      break;
fi;
printf("SetBrake(12)\n");
ACCState = BRAKING;

:: (ACCState == SAFE_ZONE) ->
printf("In State: SAFE_ZONE\n");
printf("Calc SafeRate\n");
if
:: !Detected ->
  break;
fi;
printf("SetBrakeRate(rate)\n");
ACCState = BRAKING;
od;

composite!_pid; /* Return the pid to tell the ACC the Engaged state has exited. */
}
The radar unit detects a lead vehicle; it interrupts the sleeping radar listener. The radar listener then initializes the ACC and goes back to sleep.

Once activated, the ACC gets the Cruise Control information and begins looping. First it pulls the Radar unit and the ECM for information, using the getinfo() function of Radar, and the getSpeed() function of the ECM.

Based on the new information from the Radar unit and the ECM, the ACC initializes the appropriate warnings in the GUI and calculates the appropriate deceleration rate.

The ACC sets the deceleration rate and then reloops.

When the Radar unit loses the lead car, it interrupts the Radar Listener again which tells the ACC to begin a cleanup sequence.

The ACC stops braking, returns throttle control to manual, and deactivates all warnings in the GUI. The ACC then goes back to sleep.
The radar unit detects a lead vehicle it interrupts the sleeping radar listener. The radar listener then initializes the ACC and goes back to sleep.

Once activated, the ACC gets the Cruise Control information and begins looping. First it polls the Radar unit for information, using the `getInfo()` function of Radar, and the `getSpeed()` function of the ECM.

Based on the new information from the Radar unit and the ECM, the ACC initializes the appropriate warnings in the GUI and calculates the appropriate deceleration rate.

The ACC sets the deceleration rate to maximum (12) to avoid imminent collision and then reloops.

When the Radar unit loses the lead car, it interrupts the Radar Listener again which tells the ACC to begin a cleanup sequence.

The ACC stops braking, resets the cruise control to its original speed, and deactivates all warnings in the GUI. The ACC then goes back to sleep.