Faculty of Information and Communications Technologies

Modeling an Adaptive Cruise Controller in the Architecture Analysis and Design Language: A Case Study

Technical Report: C4-01 TR M51

Stefan Björnander, Mälardalen University, Sweden
Lars Grunske, Swinburne University of Technology, Australia
30th November 2008

SWINBURNE UNIVERSITY OF TECHNOLOGY

Modeling an Adaptive Cruise Controller in the Architecture Analysis and Design Language: A Case Study

Stefan Björnander[1,2] and Lars Grunske[2]
[1] School of IDE, Mälardalen University, Box 883, 72123 Västerås, Sweden, Phone: +46 21 101 689, stefan.bjornander@mdh.se
[2] Faculty of ICT, Swinburne University of Technology Hawthorn, VIC 3122, Australia, Phone: +61 3 9214 5397, lgrunske@swin.edu.au

Abstract

An Adaptive Cruise Control (ACC) is a part of an automobile system which purpose is to control the vehicle speed with regards to the surrounding environment. The objective of this report is to evaluate whether the Architecture and Analysis Description Language (AADL) is suitable to model an ACC. This report describes an ACC modeled in AADL with its Behavior and Error Annex.

Contents

1 Introduction

A cruise control system is a rather simply system that keeps the vehicle speed at a constant level. An Adaptive Cruise Control (ACC) is an extension of a cruise control system. Its task is to adjust the speed of the vehicle with regards to other vehicles, the conditions of the road, weather conditions, etc. In figure 1, the vehicle to the left is equipped with an ACC that keeps track of the vehicle to the right. The upper vehicle is not detected by the ACC. If it should change lanes, however, it would be detected.

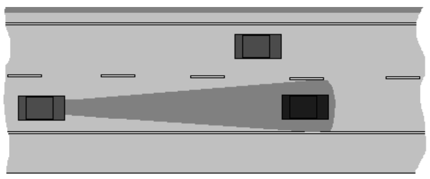


Figure 1: The left-hand car detects the right-hand car.

In 1987, the European Union's EUREKA program initiated the Prometheus project on autonomous vehicles [11]. Its purpose was to enhance traffic security by introducing different information systems for automobiles. The first ACC prototype was developed in the early nineties as part of the EUREKA program.

An ACC system has two objectives. The first one is to reduce to driver's workload and make the drive more comfortable and safe by automatically adjusting the vehicle speed with regards to obstacles, especially the vehicles ahead [2, 6]. The second objective is to avoid collisions. The commercial available ACC systems combine those two objectives.

Section 2 describes the ACC modeled in AADL. Subsection 2.1 gives an overview of AADL and subsection 2.2 describes the model as it is implemented in AADL, with its two subsystem *Console* and *Controller*. Subsection 2.3 describes the simulation of the model and subsection 2.4 describes the AADL error model of the two ACC subsystems.

2 AADL

AADL (aadl.info) is a language [4] intended for the design of both the hardware and the software of a system. It is an Society of Automotive Engineers (SAE, www.sae.org) standard and is based on MetaH [10] and UML 2.0 [1, 8, 9].

2.1 An Overview

The component abstractions of AADL are separated into three categories. The first category is the application software:

- **Thread**. A thread can execute concurrently and be organized into thread groups.
- **Thread Group**. A thread group is a component abstraction for logically organizing threads or thread groups within a process.
- **Process**. A process is a protected address space whose boundaries are enforced at runtime.
- **Data**. A data component models types and static data.
- **Subprogram**. A subprogram models a callable piece of source code.

The second category is the execution platform (the hardware):

- **Processor**. A processor schedules and executes threads.
- **Memory**. A memory component is used to store code and data.
- **Device**. A device represents sensors and actuators that interface with the external environment.
- **Bus**. A bus interconnects processors, memory, and devices.

The third category contains only one element: the system component. System components can consist of software and hardware components as well as other systems.

The components' types are defined by parameterized sets of properties. Furthermore, components communicate with each other through ports. It is possible to define physical port-to-port connections as well as logical flows through chains of ports. Component definitions are divided into component types holding the public (visible to other components) features, and component implementations that define the private (inner) parts of the component.

The AADL standard [7] includes runtime semantics for mechanisms of exchange and control of data, including message passing, event passing, synchronized access to shared components, thread scheduling protocols, and timing requirements.

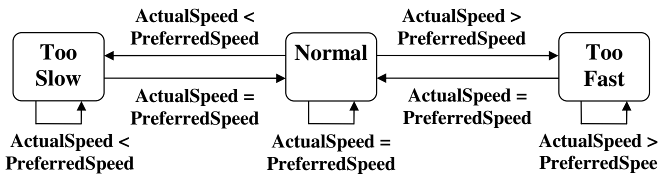
AADL can be used to model and analyze systems already in use as well as to design new systems. AADL can also be used in the analysis of partially defined architectural patterns. Moreover, AADL supports the early prediction and analysis of critical system qualities, such as performance, schedulability, and reliability.

Within the core language, property sets can be declared that add new properties for components. Additional models and properties can also be included by utilizing the extension capabilities of the language. The properties and extensions can be used to incorporate analysis at the architectural design level.

AADL components interact through defined interfaces. A component interface consists of directional flow through data ports for state data, event data ports for message data, event ports for asynchronous events, subprogram calls, and explicit access to data components. Application components have properties that specify timing requirements such as period, worst-case execution time, deadlines, space requirements, and arrival rates [5].

The AADL standard provides the possibility to extend the language with *annexes*. Currently, one of them is the Behavior Annex. It models the behavior of the system as a state machine. Figure 2 shows a high-level behavior model of an cruise control system, which purpose is to adjust the actual vehicle speed to the (by the driver) preferred speed. The model of figure 2 is implemented in listing 1.

In the state machine, SpeedUp and SpeedDown are ports connected to the throttle. An event is sent by applying a exclamation (!) mark to the port. An event is received by applying a question (?) mark to the port.



Too Slow
Normal
Too Fast
ActualSpeed < PreferredSpeed
ActualSpeed > PreferredSpeed
ActualSpeed = PreferredSpeed
ActualSpeed = PreferredSpeed
ActualSpeed < PreferredSpeed
ActualSpeed = PreferredSpeed
ActualSpeed > PreferredSpee

Figure 2: A Behavior Model of an Adaptive System.

Listing 1 A Behavior Model of an Adaptive System.

```
annex behavior_specification
{**
  states
    Normal : initial state;
    TooSlow, TooFast : state;

  state variables
    ActualSpeed, PreferredSpeed: Behaviour::Integer;

  transitions
    Normal -[on (ActualSpeed = PreferredSpeed)]-> Normal {}

    Normal -[on (ActualSpeed < PreferredSpeed)]-> TooSlow {SpeedUp!}
    TooSlow -[on (ActualSpeed < PreferredSpeed)]-> TooSlow {SpeedUp!}
    TooSlow -[on (ActualSpeed = PreferredSpeed)]-> Normal {}

    Normal -[on (ActualSpeed > PreferredSpeed)]-> TooFast {SpeedDown!}
    TooFast -[on (ActualSpeed > PreferredSpeed)]-> TooFast {SpeedDown!}
    TooFast -[on (ActualSpeed = PreferredSpeed)]-> Normal {}
**};
```

Another AADL extension is the Error Annex, which provides the capability to annotate AADL components with dependability related information. Figure 3 shows an error model for the radar lens of an radar unit. In cold climate the lens may be covered with ice, which destroys its ability to measure the distance to the obstacle. However, the error is temporary if the radar unit is equipped with a thermostat and a heater. When the lens has thawed, it is operational again. On the other hand, if the lens is hit by a stone so that the glass cracks, a permanent error has occurred. Finally, the lens will eventually become worn out.

We can also define the probability for each event to occur. The probability is given with the keyword *fixed* or *poisson*, where a fixed value denotes the probability between 0 and 1 and a poisson value λ denotes a probability of $-e^{-\lambda t}$. The fixed value is used in situations where the event probability is randomly distributed, such as the broken lens probability. The poisson value is used in situations where the probability increases over time, such as the worn out probability. The model of figure 3 is implemented in listing 2.

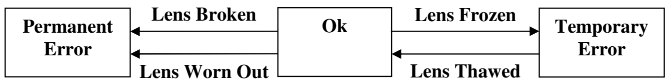


Figure 3: An Error Model of the Radar Lens.

Listing 2 An Error Model of the Radar Lens.

```
annex error_specification
{**
  error model LensError
    features
      Ok: initial error state;
      PermanentError, TemporaryError: state;
      RadarLensBroken, RadarLensFrozen,
      RadarLensThawed, RadarLensWornOut: error event;
  end LensError;

  error model implementation LensError.impl
    transitions
      Ok-[RadarLensBroken]->PermanentError;
      Ok-[RadarLensWornOut]->PermanentError;
      Ok-[RadarLensFrozen]->TemporaryError;
      TemporaryError[RadarLensThawed]->Ok;
    properties
      Occurance => poisson 0.001 applies to RadarLensWornOut;
      Occurance => fixed 0.01 applies to RadarLensBroken;
      Occurance => fixed 0.1 applies to RadarLensFrozen;
      Occurance => fixed 0.9 applies to RadarLensThawed;
  end LensError.impl;
**};
```

2.2 The Model

The ACC system in this model comprises the two subsystems *Controller* and *Console*, which communicate by a *Lan* bus (see figure 4).

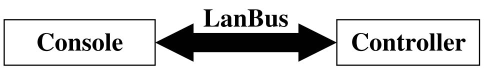


Figure 4: The ACC System.

2.2.1 The Console System

The console subsystem consists of a processor, a RAM memory with a memory bus, the panel and display devices, and a process that runs threads that sample the input from the panel, calculate the speed, and send the speed to the display (see figure 5 and listing 3). Its task is to receive instructions from the driver concerning the preferred speed and displaying it to the driver on the dashboard. The panel device is equipped with an on/off button and two buttons marked *plus* and *minus*. The driver can adjust the preferred speed and turn the ACC system on or off by pressing the buttons. The panel device notifies the console process, which delegates the calculation of the new preferred speed to the sampling and computation threads (see section 2.2.2).

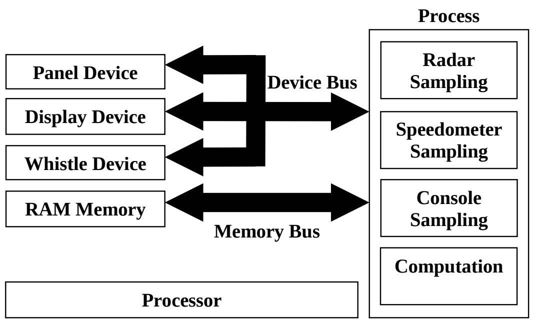


Figure 5: The Console Subsystem.

2.2.2 The Controller System

The controller system is constituted by a processor, a RAM memory with a memory bus, the radar, steering wheel, and speedometer devises as well as break and throttle actuators and a device bus. The processor runs four threads: three threads that samples the input data from the radar and speedometer devices as well as the console system. The fourth thread of

Listing 3 The Console System implemented in AADL.

```
system ConsoleSystem
  features
    outToggle: out event port;
    outData: out data port AccTypes::Stream;
    memoryBus: requires bus access MemoryBus;
    deviceBus: requires bus access DeviceBus;
    lanBus: provides bus access LanBus;
end ConsoleSystem;

system implementation ConsoleSystem.impl
  subcomponents
    consoleProcess: process ConsoleProcess.impl;
    consoleProcessor: processor ConsoleProcessor.impl;
    ramMemory: memory RamMemory;
    panelDevice: device PanelDevice;
  connections
    event port panelDevice.outToggle -> outToggle;
    data port consoleProcess.outSpeed -> outData;
    event port panelDevice.outPlus -> consoleProcess.inPlus;
    event port panelDevice.outMinus -> consoleProcess.inMinus;
  properties
    Actual_Processor_Binding => reference consoleProcessor
                                 applies to consoleProcess;
    Actual_Memory_Binding => reference ramMemory
                              applies to consoleProcess;
end ConsoleSystem.impl;
```

the process calculates the proper degree of break and throttle application and notifies the break and throttle actuators, respectively (see figure 6 and listing 4).

The radar device is located at the front of the car. Besides the radar itself, the device also holds a unit capable of moving the line of sight in the horizontal direction. This is necessary in order to aim the radar beam correctly in curves. The radar measures the distance to the nearest obstacle and sends the information to the computation unit.

The radar sends two signals to the controller process: a boolean value which is true if an obstacle has been detected and, in that case, the distance between the vehicle and the obstacle. The speedometer sends the current vehicle speed to the process. The steering wheel measures the curvature of the vehicle and sends the information directly to the radar in order to adjust the radar beam to the curvature of the road.

As the information sent by the device bus, it can only be read and written one byte at a time. Therefore, there are threads reading (sampling) the bus information from the radar and speedometer.

2.3 Implementation

The computation unit of the controller subsystem saves the previous distance to the nearest obstacle and compares it to the current distance in order to

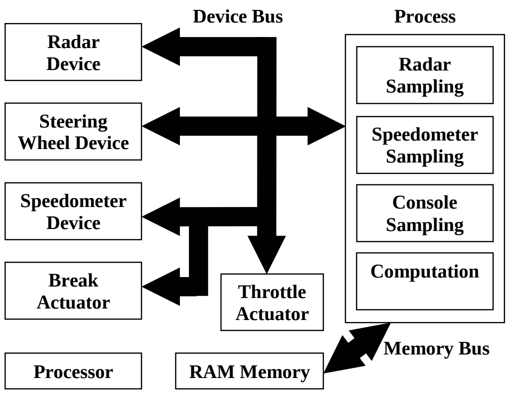


Figure 6: The Controller Subsystem.

determine the relative speed between the vehicle and the obstacle. That information is used in several ways. If the distance has decreased, the brakes are applied in order to slow down the vehicle. If the distance has increased, the *electronic throttle system* is engaged in order to increase the vehicle speed. If the distance has decreased radically and the obstacle has entered an unsafe range (normally 10 meters), the system applies full brakes in order to stop the vehicle. If no obstacle is detected, the vehicle speed is adjusted to the speed preferred by the driver.

2.3.1 The Theoretical Foundation

In this model, a *vehicle* follows an *obstacle* (normally another vehicle). The main point of this section is to make sure that when an obstacle is detected, the vehicle shall be able to break enough to avoid a collision. Informally, the task is performed by reducing the speed with a constant deceleration (negative acceleration) so that the vehicle speed is equal to the obstacle speed (the relative speed is zero).

In the trivial case when the vehicle and the obstacle travel at constant speed, that action would be enough to avoid a collision. However, in the general case when the vehicle and the obstacle vary their speeds and accelerations, the distance between them must be continuously monitored by the ACC in order to properly adjust the vehicle speed.

This section introduces the formal definitions and theorems necessary to establish that the ACC will make sure that the vehicle avoids a collision.

The vehicle speed, measured by the speedometer device, is denoted v_v and the obstacle distance (the distance between the vehicle and the obstacle), measured by the radar device, is denoted d. By comparing the current vehicle speed and obstacle distance with the previous vehicle speed $v_v{}'$ and the previous distance d', the obstacle speed $v_o = v_v - v_v{}' + \frac{d-d'}{T}$ (T is the

Listing 4 The Controller System implemented in AADL.

```
system ControllerSystem
  features
    inConsoleToggle: in event port;
    inConsoleData: in data port AccTypes::Stream;
    LanBus: requires bus access LanBus;
end ControllerSystem;

system implementation ControllerSystem.impl
  subcomponents
    computeProcess: process ControllerProcess.impl;
    computeProcessor: processor ControllerProcessor.impl;
    ramMemory: memory RamMemory;
    deviceBus: bus DeviceBus;
    memoryBus: bus memoryBus;
    radarDevice: device RadarDevice;
    speedomoterDevice: device SpeedometerDevice;
    wheelDevice: device SteeringWheelDevice;
    brakeDevice: device BrakeDevice;
    throttleDevice: device ThrottleDevice;
  connections
    bus access memoryBus -> computeProcessor.memoryBus;
    bus access memoryBus -> ramMemory.memoryBus;
    bus access deviceBus -> computeProcessor.deviceBus;
    bus access deviceBus -> radarDevice.deviceBus;
    bus access deviceBus -> speedomoterDevice.deviceBus;
    bus access deviceBus -> wheelDevice.deviceBus;
    bus access deviceBus -> brakeDevice.deviceBus;
    bus access deviceBus -> throttleDevice.deviceBus;
    event port inConsoleToggle -> computeProcess.inConsoleToggle;
    data port inConsoleData -> computeProcess.inConsoleData;
    event port radarDevice.outError -> computeProcess.inRadarError
               {Actual_Connection_Binding => reference deviceBus;};
    data port radarDevice.outData -> computeProcess.inRadarData
              {Actual_Connection_Binding => reference deviceBus;};
    data port speedomoterDevice.outData ->
              computeProcess.inSpeedometerData
              {Actual_Connection_Binding => reference deviceBus;};
    data port wheelDevice.outData -> radarDevice.inData
              {Actual_Connection_Binding => reference deviceBus;};
    event port computeProcess.outBrake -> brakeDevice.inBrake;
    event port computeProcess.outThrottle -> throttleDevice.inThrottle;
end ControllerSystem.impl;
```

radar device period) and the relative speed between the vehicle and the obstacle $v_r = v_o - v_v$ is calculated.

In the same manner, the vehicle acceleration $a_v = \frac{v_v - v_v'}{T}$ and obstacle acceleration $a_o = \frac{v_o - v_o'}{T}$ as well as their relative acceleration $a_r = a_o - a_v$ is calculated (see figure 7).

Definition 1. *Given a vehicle and an obstacle with distance, speeds, and accelerations as above, the function* ***brake*** *calculating the deceleration necessary for breaking the vehicle in order to avoid a collision is given by:*

$$brake\,(d, v_r, a_r) = a_r - \frac{v_r^2}{d}$$

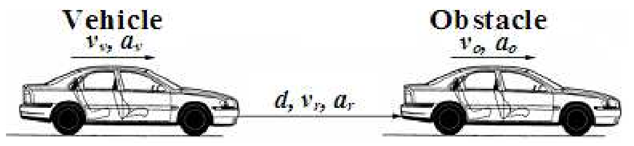


Figure 7: The vehicle and obstacle speed and acceleration.

Definition 2. *Given a vehicle and an obstacle with distance, speeds, and accelerations as above and with breaking deceleration a_b, $stop\,(d, v_r, a_r, a_b)$ is true if the vehicle is able to slow down enough to avoid colliding with the obstacle.*

Theorem 1. *$stop\,(d, v_r, a_r, a_b)$ (A) is true iff $\frac{v_r^2}{a_r - a_b} \leq d$ (B).*

Proof. $A \Rightarrow B$: If $stop\,(d, v_r, a_r, a_b)$ is true, then the distance d_n needed to slow down the vehicle speed to the obstacle speed with constant (negative) acceleration $a_r - a_b$ must be less or equal to the distance d between the vehicle and the obstacle. The distance is given by:

$$v_r = (a_r - a_b)\,t \Rightarrow t = \frac{v_r}{a_r - a_b}$$

$$d_n = v_r t = \frac{v_r^2}{a_r - a_b}$$

This gives that if $stop\,(d, v_r, a_r, a_b)$ is true then

$$d_n = \frac{v_r^2}{a_r - a_b} \leq d$$

$B \Rightarrow A$: The right hand side expression $\frac{v_r^2}{a_r + a_b}$ denotes the distance d_n needed to slow down the vehicle speed to the obstacle's speed with constant (negative) acceleration $a_r + a_b$. If it less or equal to the distance d between the vehicle and the obstacle, the vehicle manages to slow down enough to avoid a collision and $stop\,(d, v_r, a_r, a_b)$ is true. □

Theorem 2. *$stop\,(d, v_r, a_r, a_b)$ (A) is true iff $brake\,(d, v_r, a_r) \geq a_b$ (B).*

Proof. $A \Rightarrow B$: $stop\,(d, v_r, a_r, a_b)$ is true $\Rightarrow \frac{v_r^2}{a_r - a_b} \leq d \Rightarrow \frac{v_r^2}{d} \leq a_r - a_b \Rightarrow a_b \leq a_r - \frac{v_r^2}{d} = brake\,(d, v_r, a_r)$.

$B \Rightarrow A$: $brake\,(d, v_r, a_r) \geq a_b \Rightarrow \frac{v_r^2}{a_r + brake(d, v_r, a_r)} \leq \frac{v_r^2}{a_r + a_b} \leq d \Rightarrow$ $stop\,(d, v_r, a_r, a_b)$ is true. □

Theorem 3. *$stop\,(d, v_r, a_r, brake\,(d, v_r, a_r))$ is true.*

Proof. Definition 1 defined *break* as $a_r - \frac{v_r^2}{d}$ and theorem 2 implies that $stop\left(d, v_r, a_r, brake\left(d, v_r, a_r\right)\right)$ is true if $\frac{v_r^2}{a_r - brake(d, v_r, a_r)} \leq d$. This gives that the assumption is true if $\frac{v_r^2}{a_r - \left(a_r - \frac{v_r^2}{d}\right)} \leq d$:

$$\frac{v_r^2}{a_r - \left(a_r - \frac{v_r^2}{d}\right)} = \frac{v_r^2}{a_r - a_r + \frac{v_r^2}{d}} = \frac{v_r^2}{\frac{v_r^2}{d}} = d \leq d$$

□

2.3.2 The ACC Algorithm

In this section, algorithm 1 is introduced. By keeping track of the distance, speeds, and accelerations of the vehicle and the obstacle as well as applying the function *break* from section 2.3.1 every time the radar device detects an obstacle, a collision is avoided in accordance to theorem 3.

If the radar device does not detect an obstacle, it instead strives to adjust the vehicle speed in accordance with the speed preferred by the driver.

2.3.3 The Behavior Model

As part of the case study, the behavior of the ACC is implemented as an abstract state machine with the AADL Behavior Annex (see listing 5).

The model has five states:

- **Normal**. No obstacle is detected and the vehicle is travel at the (by the driver) preferred speed.
- **Too Slow**. No obstacle is detected but the vehicle is travel at a speed lower than the (by the driver) preferred speed. The throttle is applied in order to increase the speed.
- **Too Fast**. No obstacle is detected but the vehicle is travel at a speed higher than the (by the driver) preferred speed. The throttle is reduced in order to decrease the speed.
- **Detected**. An obstacle is detected and the vehicle is adjusting its speed in accordance to the distance to the obstacle and their relative speed by applying algorithm 1.
- **Emergency**. This state is included as a precaution. If the obstacle appears inside a certain unsafe range (normally 10 meters from the vehicle), full brakes is applied in order to avoid a collision.

When an obstacle is detected, the idea is that the vehicle immediate shall adjust its speed. As it in practice is very hard to calculate a suitable reduction of the speed, the speed adjusting process is modeled as a feed-back system.

Algorithm 1 The ACC Algorithm

```
Require:
  $f$: The radar's frequency (the number of times per second it reports the distance).
  $T$: The radar period, the time interval between the radar readings ($T = 1/f$).
  $d, v_v, a_v, v_o, a_o, v_v{}', v_o{}'$: Variables defined in section 2.3.1.
  $break$: The function defined in definition 1.
  $a_{acc}, a_{dec}$: The vehicle's maximal acceleration and deceleration (both values are positive).

  $v_v{}' \leftarrow 0$
  $v_o{}' \leftarrow 0$

  loop
    $v_v \leftarrow Speedometer.Speed()$

    if $Radar.Detected()$ then
      $d \leftarrow Radar.Distance()$

      if $d \leq SecurityDistance$ then
        /*An obstacle has been detected inside the unsafe range, and we apply full brake.*/
        $Break.ApplyFull()$
      else
        /*An obstacle has been detected, the speed is reduced in order to avoid a collision.*/
        $a_v \leftarrow \frac{v_v - v_v{}'}{T}$
        $v_o \leftarrow v_v - v_v{}' + \frac{d - d'}{T}$
        $a_o \leftarrow \frac{v_o - v_o{}'}{T}$

        /*We reduce the throttle in proportion as the deceleration.*/
        $Throttle.Reduce\left(break\left(d, v_o - v_v, a_o - a_v\right)\right)$

        /*Finally, we save the previous values.*/
        $v_v{}' \leftarrow v_v$
        $v_o{}' \leftarrow v_o$
      end if
    else if $DriverConsole.IsOn()$ then
      /*If no obstacle has been detected, the speed is adjusted to the (by the driver) preferred speed.*/
      $v_p \leftarrow DriverConsole.PreferredSpeed()$

      if $v_v < v_p$ then
        $a_p \leftarrow \frac{v_p - v_v}{T}$
        $Throttle.Apply\left(\min\left(a_{acc}, a_p\right)\right)$
      else if $v_v > v_p$ then
        $a_p \leftarrow \frac{v_v - v_p}{T}$
        $Throttle.Reduce\left(\min\left(a_{dec}, a_p\right)\right)$
      else
        /*If the vehicle travels at preferred speed with no obstacle detected, no action is taken.*/
      end if
    end if
  end loop
```

Listing 5 A Behavior Model of an Adaptive System.

```
annex behavior_specification
{**
  states
    Normal : initial state;
    Detected, TooSlow, TooFast, Emergency: state;

  transitions
    Normal -[on (ObstacleDetected?) and
                (ObstacleDistance? < UNSAFE_RANGE)]-> Emergency {}
    TooFast -[on (ObstacleDetected?) and
                 (ObstacleDistance? < UNSAFE_RANGE)]-> Emergency {}
    TooSlow -[on (ObstacleDetected?) and
                 (ObstacleDistance? < UNSAFE_RANGE)]-> Emergency {}
    Emergency -[on (ObstacleDetected?) and (ObstacleDistance? <
                   UNSAFE_RANGE)]-> Emergency {FullBreak!}
    Emergency -[on (ActualSpeed = 0)]-> Normal {}

    Normal -[on (ObstacleDetected?) and
                (ActualSpeed? > ObstacleSpeed?)]-> Detected {}
    TooFast -[on (ObstacleDetected?) and
                 (ActualSpeed > ObstacleSpeed?)]-> Detected {}
    TooSlow -[on (ObstacleDetected?) and
                 (ActualSpeed > ObstacleSpeed?)]-> Detected {}
    Detected -[on (ObstacleDetected?) and
                  (ActualSpeed? > ObstacleSpeed?)]->Detected{SpeedDown!}

    Detected -[on (not ObstacleDetected?) and (ConsoleOn?) and
                  (PreferredSpeed? > ActualSpeed?)]-> TooFast {}
    Detected -[on (not ObstacleDetected?) and (ConsoleOn?) and
                  (PreferredSpeed? < ActualSpeed?)]-> TooSlow {}
    Detected -[on !(ObstacleDetected?) and ((not ConsoleOn?) or
                   (PreferredSpeed? = ActualSpeed?))]-> Normal {}

    Normal -[on (ConsoleOn?) and
                (ActualSpeed? < PreferredSpeed?)]-> TooSlow {}
    TooSlow -[on (ConsoleOn?) and
                 (ActualSpeed? > PreferredSpeed?)]-> TooFast {}
    TooSlow -[on (not ConsoleOn?) or
                 (ActualSpeed? = PreferredSpeed?)]-> Normal {}
    TooSlow -[on (ConsoleOn?) and
                 (ActualSpeed? < PreferredSpeed?)]-> TooSlow{SpeedUp!}

    Normal -[on (ConsoleOn?) and
                (ActualSpeed? > PreferredSpeed?)]-> TooFast {}
    TooFast -[on (ConsoleOn?) and
                 (ActualSpeed? < PreferredSpeed?)]-> TooSlow {}
    TooFast -[on (not ConsoleOn?) or
                 (ActualSpeed? = PreferredSpeed?)]-> Normal {}
    TooFast -[on (ConsoleOn?) and
                 (ActualSpeed? > PreferredSpeed?)]-> TooFast{SpeedDown!}

    Normal ->[on !(ObstacleDetected?) and ((not ConsoleOn?) or
                  (PreferedSpeed? = ActualSpeed?))] -> Normal {}
  **};
```

When the machine is in the detected state, the speed is reduced. Should the vehicle come to close to the obstacle, so it enters the unsafe range, full brakes is applied until the vehicle is stationary.

2.4 The Error Model

In addition to the behavior model, there is also an error model [3] in AADL. Each of the two subsystems Console and Controller has its own error handling.

2.4.1 The Console System

The point of the Console error model is that it shall work even though one or several components malfunction. The model has four states:

- **Ok**. All parts of the system works.
- **PanelDown**. The panel has malfunctioned.
- **DisplayDown**. The display has malfunctioned.
- **PanelDisplayDown**. The panel and the display have malfunctioned.

Listing 6 An Error Model of the Console System.

```
annex error_specification
{**
  error model ConsoleError
    features
      Ok: initial error state;
      PanelDown, DisplayDown, PanelDisplayDown: error state;
      PanelFailure, DisplayFailure : error event;
  end ConsoleError;

  error model implementation ConsoleError.impl
    transitions
      Ok-[PanelFailure]->PanelDown;
      DisplayDown-[PanelFailure]->PanelDisplayDown;
      Ok-[DisplayFailure]->DisplayDown;
      PanelDown-[DisplayFailure]->PanelDisplayDown;
    properties
      Occurance => fixed 0.01 applies to PanelFailure;
      Occurance => fixed 0.1 applies to DisplayFailure;
  end ConsoleError.impl;
**};
```

2.4.2 The Controller System

The only component that can be expected to work again after it malfunction is the radar, since its lens may be frozen (see figure 3 and listing 2). Therefore, in order to provide the model with a way to return to an error free state regarding the radar device, the error state RadarUp is included. One peculiar thing about the Error Annex is that even thought RadarUp is an error-free state, it is still modeled as an error state.

The first transition looks up the last valid detected notice from the radar. If an obstacle was detected during the previous scanning cycle and the radar malfunctions during the current cycle, the vehicle is immediate stopped. If no obstacle was detected, only a warning signal is given and the ACC is transferred to manual drive. A similar action is taken if the speedometer malfunctions.

If the throttle malfunctions the vehicle is stopped and if the brakes malfunctions the vehicle is stopped by the throttle.

- **Ok**. All system works.
- **RadarDown**. The radar is down but the speedometer works.
- **SpeedDown**. The speedometer is down but the radar works.
- **RadarSpeedDown**. The radar and speedometer are down.
- **FailSafe**. Such a serious error has occurred that the vehicle has been completely stopped.

3 Conclusions

In the model of this paper, almost all of the AADL features has been applied. The *data* features is used to define the data that is sent over the *buses*. The model is constituted by the two *systems* Console and Controller. Each system consists of several *devices* and *processor* with a *memory* and a *process* divided into several *threads*. However, *thread groups* and *subprograms* are not represented in this model.

The support for both software and hardware components entails that AADL is a powerful language suitable for modeling and reason about large systems. Moreover, the behavioral annex adds logical power while the error annex adds dependability to the system model.

A Source Code

```
package AccTypes
  public
    data Integer
      properties
        Source_Data_Size => 32 bits;
    end Integer;

    subprogram SetInteger
      features
        Input: in parameter Integer;
    end SetInteger;

    subprogram GetInteger
      features
        Output: out parameter Integer;
    end GetInteger;
```

Listing 7 An Error Model of the Controller System.

```
annex error_specification
{**
  error model ControllerError
    features
      Ok: initial error state;
      RadarDown, SpeedDown, RadarSpeedDown, FailSafe: error state;
      RadarFailure, RadarWorks, SpeedometerFailure,
      BreaksFailure, ThrottleFailure: error event;
  end ControllerError;

  error model implementation ControllerError.impl
    transitions
      Ok-[ObstacleDetected? and RadarFailure]->FailSafe {FullBrake!}
      Ok-[not ObstacleDetected? and RadarFailure]->NoRadarDrive{Whistle!}
      NoSpeedDrive-[ObstacleDetected? and RadarFailure]->
      FailSafe {FullBrake!}
      NoSpeedDrive-[not ObstacleDetected? and RadarFailure]->
      NoRadarSpeedDrive {Whistle!}

      Ok-[SpeedometerDown]->NoSpeedDrive {Whistle!}
      RadarDown-[SpeedometerDown]->NoRadarSpeedDrive {Whistle!}

      NoRadarDrive-[RadarWorks]->Ok.
      NoRadarSpeedDrive-[RadarWorks]->NoSpeedDrive.

      Ok-[BreaksDown]->FailSafe {FullReducedThrottle!}
      Ok-[ThrottleDown]->FailSafe {FullBrake!}

    properties
      Occurance => fixed 0.5 applies to RadarWorks;
      Occurance => fixed 0.1 applies to RadarFailure;
      Occurance => fixed 0.05 applies to SpeedometerFailure;
      Occurance => fixed 0.01 applies to ThrottleFailure;
      Occurance => fixed 0.001 applies to BreaksFailure;
  end ControllerError.impl;
**};
```

```
    data Float
      properties
        Source_Data_Size => 64 bits;
    end Float;

    subprogram SetFloat
      features
        Input: in parameter Float;
    end SetFloat;

    subprogram GetFloat
      features
        Output: out parameter Float;
    end GetFloat;

    data Boolean
      properties
        Source_Data_Size => 8 bits;
    end Boolean;
    data VehicleInfo
      features
        SetPreviousVehicleSpeed: subprogram SetInteger;
        GetPreviousVehicleSpeed: subprogram GetInteger;
    end VehicleInfo;

    data implementation VehicleInfo.impl
      subcomponents
        PreviousVehicleSpeed: data Integer;
    end VehicleInfo.impl;

    data Stream
    end Stream;

    data implementation Stream.impl
      properties
        Source_Data_Size => 8bits;
    end Stream.impl;
end AccTypes;

bus MemoryBus
end MemoryBus;

bus DeviceBus
end DeviceBus;

bus LanBus
end LanBus;

memory RamMemory
  features
    memoryBus: requires bus access MemoryBus;
end RamMemory;

memory implementation RamMemory.impl
  properties
    Word_Size => 8bits;
end RamMemory.impl;

processor ConsoleProcessor
  features
    memoryBus: requires bus access MemoryBus;
    deviceBus: requires bus access DeviceBus;
end ConsoleProcessor;

processor implementation ConsoleProcessor.impl
  properties
    Scheduling_Protocol => (RMS, EDF, Sporadicserver, SlackServer, ARINC653);
end ConsoleProcessor.impl;

thread ComputeSpeedThread
  features
    inPlus: in event port;
    inMinus: in event port;
    outSpeedDisplay: out data port AccTypes::Integer;
    outSpeedSend: out data port AccTypes::Integer;
end ComputeSpeedThread;

thread implementation ComputeSpeedThread.impl
  properties
    Period => 100ms;
end ComputeSpeedThread.impl;
```

```
thread SendSpeedThread
  features
    inSpeed: in data port AccTypes::Integer;
    outSpeed: out data port AccTypes::Stream;
end SendSpeedThread;

thread implementation SendSpeedThread.impl
  properties
    Period => 100ms;
end SendSpeedThread.impl;

thread DisplaySpeedThread
  features
    inSpeed: in data port AccTypes::Integer;
end DisplaySpeedThread;

thread implementation DisplaySpeedThread.impl
  properties
    Period => 100ms;
end DisplaySpeedThread.impl;

process ConsoleProcess
  features
    inPlus: in event port;
    inMinus: in event port;
    outSpeed: out data port AccTypes::Stream;
  annex error_specification
  {**
    error model ConsoleError
      features
        Ok: initial error state;
        PanelDown, DisplayDown, PanelDisplayDown: error state;
        PanelFailure, DisplayFailure : error event;
    end ConsoleError;

    error model implementation ConsoleError.impl
      transitions
        Ok-[PanelFailure]->PanelDown;
        DisplayDown-[PanelFailure]->PanelDisplayDown;
        Ok-[DisplayFailure]->DisplayDown;
        PanelDown-[DisplayFailure]->PanelDisplayDown;
      properties
        Occurance => fixed 0.01 applies to PanelFailure;
        Occurance => fixed 0.1 applies to DisplayFailure;
    end ConsoleError.impl;
  **};
end ConsoleProcess;

process implementation ConsoleProcess.impl
  subcomponents
    computeSpeedThread: thread ComputeSpeedThread.impl;
    sendSpeedThread: thread SendSpeedThread.impl;
    displaySpeedThread: thread DisplaySpeedThread.impl;
  connections
    event port inPlus -> computeSpeedThread.inPlus;
    event port inMinus -> computeSpeedThread.inMinus;
    data port computeSpeedThread.outSpeedDisplay -> displaySpeedThread.inSpeed;
    data port computeSpeedThread.outSpeedSend -> sendSpeedThread.inSpeed;
    data port sendSpeedThread.outSpeed -> outSpeed;
end ConsoleProcess.impl;
```

```
device PanelDevice
  features
    outPlus: out event port;
    outMinus: out event port;
    outToggle: out event port;
end PanelDevice;

system ConsoleSystem
  features
    outToggle: out event port;
    outData: out data port AccTypes::Stream;
    memoryBus: requires bus access MemoryBus;
    deviceBus: requires bus access DeviceBus;
    lanBus: provides bus access LanBus;
end ConsoleSystem;

system implementation ConsoleSystem.impl
  subcomponents
    consoleProcess: process ConsoleProcess.impl;
    consoleProcessor: processor ConsoleProcessor.impl;
    ramMemory: memory RamMemory;
    panelDevice: device PanelDevice;
  connections
    event port panelDevice.outToggle -> outToggle;
    data port consoleProcess.outSpeed -> outData;
    event port panelDevice.outPlus -> consoleProcess.inPlus;
    event port panelDevice.outMinus -> consoleProcess.inMinus;
  properties
    Actual_Processor_Binding => reference consoleProcessor
                                applies to consoleProcess;
    Actual_Memory_Binding => reference ramMemory
                             applies to consoleProcess;
end ConsoleSystem.impl;

device SpeedometerDevice
  features
    outData: out data port AccTypes::Stream;
    deviceBus: requires bus access DeviceBus;
end SpeedometerDevice;

device SteeringWheelDevice
  features
    outData: out data port AccTypes::Stream;
    deviceBus: requires bus access DeviceBus;
end SteeringWheelDevice;

property set RadarProperties is
  RangeType: type aadlinteger units (m);
  RadarRange: RadarProperties::RangeType applies to (device RadarDevice);
end RadarProperties;

device RadarDevice
  features
    outData: out data port AccTypes::Stream; -- Detected, Distance
    inData: in data port AccTypes::Stream; -- Radar Angle
    outError: out event port;
    deviceBus: requires bus access DeviceBus;
end RadarDevice;

device implementation RadarDevice.impl
  properties
    RadarProperties::RadarRange => 200m;
```

```
end RadarDevice.impl;

device BrakeDevice
  features
    inBrake: in event port;
    deviceBus: requires bus access DeviceBus;
end BrakeDevice;

device ThrottleDevice
  features
    inThrottle: in event port;
    deviceBus: requires bus access DeviceBus;
end ThrottleDevice;

thread RadarSampleThread
  features
    inData: in data port AccTypes::Stream;
    outObstacleDetected: out data port AccTypes::Boolean;
    outObstacleDistance: out data port AccTypes::Integer;
end RadarSampleThread;

thread implementation RadarSampleThread.impl
  properties
    Period => 100ms;
end RadarSampleThread.impl;

thread SpeedometerSampleThread
  features
    inData: in data port AccTypes::Stream;
    outActualSpeed: out data port AccTypes::Integer;
end SpeedometerSampleThread;

thread implementation SpeedometerSampleThread.impl
  properties
    Period => 200ms;
end SpeedometerSampleThread.impl;

thread ConsoleSampleThread
  features
    inData: in data port AccTypes::Stream;
    outPreferredSpeed: out data port AccTypes::Integer;
end ConsoleSampleThread;

thread implementation ConsoleSampleThread.impl
  properties
    Period => 200ms;
end ConsoleSampleThread.impl;

thread ComputeActionThread
  features
    inObstacleDetected: in data port AccTypes::Boolean;
    inObstacleDistance: in data port AccTypes::Integer;
    inPreferredSpeed: in data port AccTypes::Integer;
    inActualSpeed: in data port AccTypes::Integer;
    outBrake: out event port;
    outThrottle: out event port;
  annex behavior_specification
  {**
    states
      Normal : initial state;
      Detected, TooSlow, TooFast, Emergency: state;
    transitions
      Normal -[on (ObstacleDetected?) and
                  (ObstacleDistance? < UNSAFE_RANGE)]-> Emergency {}
      TooFast -[on (ObstacleDetected?) and
                   (ObstacleDistance? < UNSAFE_RANGE)]-> Emergency {}
      TooSlow -[on (ObstacleDetected?) and
                   (ObstacleDistance? < UNSAFE_RANGE)]-> Emergency {}
      Emergency -[on (ObstacleDetected?) and (ObstacleDistance? <
                     UNSAFE_RANGE)]-> Emergency {FullBreak!}
      Emergency -[on (ActualSpeed = 0)]-> Normal {}
      Normal -[on (ObstacleDetected?) and
                  (ActualSpeed? > ObstacleSpeed?)]-> Detected {}
      TooFast -[on (ObstacleDetected?) and
                 (ActualSpeed > ObstacleSpeed?)]-> Detected {}
      TooSlow -[on (ObstacleDetected?) and
                   (ActualSpeed > ObstacleSpeed?)]-> Detected {}
      Detected -[on (ObstacleDetected?) and
                     (ActualSpeed? > ObstacleSpeed?)]->Detected{SpeedDown!}
      Detected -[on (not ObstacleDetected?) and (ConsoleOn?) and
                    (PreferredSpeed? > ActualSpeed?)]-> TooFast {}
      Detected -[on (not ObstacleDetected?) and (ConsoleOn?) and
                    (PreferredSpeed? < ActualSpeed?)]-> TooSlow {}
      Detected -[on !(ObstacleDetected?) and ((not ConsoleOn?) or
                     (PreferredSpeed? = ActualSpeed?))]-> Normal {}
      Normal -[on (ConsoleOn?) and
                  (ActualSpeed? < PreferredSpeed?)]-> TooSlow {}
      TooSlow -[on (ConsoleOn?) and
                   (ActualSpeed? > PreferredSpeed?)]-> TooFast {}
      TooSlow -[on (not ConsoleOn?) or
                   (ActualSpeed? = PreferredSpeed?)]-> Normal {}
      TooSlow -[on (ConsoleOn?) and
                   (ActualSpeed? < PreferredSpeed?)]-> TooSlow{SpeedUp!}
      Normal -[on (ConsoleOn?) and
                  (ActualSpeed? > PreferredSpeed?)]-> TooFast {}
      TooFast -[on (ConsoleOn?) and
                   (ActualSpeed? < PreferredSpeed?)]-> TooSlow {}
      TooFast -[on (not ConsoleOn?) or
                   (ActualSpeed? = PreferredSpeed?)]-> Normal {}
      TooFast -[on (ConsoleOn?) and
                   (ActualSpeed? > PreferredSpeed?)]-> TooFast{SpeedDown!}
      Normal ->[on !(ObstacleDetected?) and ((not ConsoleOn?) or
                    (PreferedSpeed? = ActualSpeed?))] -> Normal {}
    **};
end ComputeActionThread;

thread implementation ComputeActionThread.impl
  subcomponents
    PreviousVehicleSpeed: data AccTypes::Integer;
    PreviousDistance: data AccTypes::Integer;
  properties
    Period => 50ms;
end ComputeActionThread.impl;

process ControllerProcess
  features
    inRadarError: in event port;
    inRadarData: in data port AccTypes::Stream;
    inConsoleToggle: in event port;
    inConsoleData: in data port AccTypes::Stream;
    inSpeedometerData: in data port AccTypes::Stream;
    outBrake: out event port;
    outThrottle: out event port;
  annex error_specification
  {**
    error model ControllerError
      features
        Ok: initial error state;
        RadarDown, SpeedDown, RadarSpeedDown, FailSafe: error state;
        RadarFailure, RadarWorks, SpeedometerFailure,
        BreaksFailure, ThrottleFailure: error event;
    end ControllerError;

    error model implementation ControllerError.impl
      transitions
        Ok-[ObstacleDetected? and RadarFailure]->FailSafe {FullBrake!}
        Ok-[not ObstacleDetected? and RadarFailure]->NoRadarDrive{Whistle!}
        NoSpeedDrive-[ObstacleDetected? and RadarFailure]->
        FailSafe {FullBrake!}
        NoSpeedDrive-[not ObstacleDetected? and RadarFailure]->
        NoRadarSpeedDrive {Whistle!}

        Ok-[SpeedometerDown]->NoSpeedDrive {Whistle!}
        RadarDown-[SpeedometerDown]->NoRadarSpeedDrive {Whistle!}

        NoRadarDrive-[RadarWorks]->Ok.
        NoRadarSpeedDrive-[RadarWorks]->NoSpeedDrive.

        Ok-[BreaksDown]->FailSafe {FullReducedThrottle!}
        Ok-[ThrottleDown]->FailSafe {FullBrake!}

      properties
        Occurance => fixed 0.5 applies to RadarWorks;
        Occurance => fixed 0.1 applies to RadarFailure;
        Occurance => fixed 0.05 applies to SpeedometerFailure;
        Occurance => fixed 0.01 applies to ThrottleFailure;
        Occurance => fixed 0.001 applies to BreaksFailure;
    end ControllerError.impl;
  **};
end ControllerProcess;

process implementation ControllerProcess.impl
  subcomponents
    radarSampleThread: thread RadarSampleThread in modes
                       (RADAR_UP_CONSOLE_UP, RADAR_UP_CONSOLE_DOWN);
    consoleSampleThread: thread ConsoleSampleThread in modes
                         (RADAR_UP_CONSOLE_UP, RADAR_DOWN_CONSOLE_UP);
    speedometerSampleThread: thread SpeedometerSampleThread in modes (RADAR_UP_CONSOLE_UP,
                             RADAR_UP_CONSOLE_DOWN, RADAR_DOWN_CONSOLE_UP);
    computeActionThread: thread ComputeActionThread in modes (RADAR_UP_CONSOLE_UP,
                         RADAR_UP_CONSOLE_DOWN, RADAR_DOWN_CONSOLE_UP);
  connections
    data port inRadarData -> radarSampleThread.inData in modes
              (RADAR_UP_CONSOLE_UP, RADAR_UP_CONSOLE_DOWN);
    data port radarSampleThread.outObstacleDetected ->
              computeActionThread.inObstacleDetected in modes
              (RADAR_UP_CONSOLE_UP, RADAR_UP_CONSOLE_DOWN);
    data port radarSampleThread.outObstacleDistance ->
              computeActionThread.inObstacleDistance in modes
              (RADAR_UP_CONSOLE_UP, RADAR_UP_CONSOLE_DOWN);
    data port inConsoleData -> consoleSampleThread.inData in modes
              (RADAR_UP_CONSOLE_UP, RADAR_DOWN_CONSOLE_UP);
    data port consoleSampleThread.outPreferredSpeed ->
              computeActionThread.inPreferredSpeed in modes
              (RADAR_UP_CONSOLE_UP, RADAR_DOWN_CONSOLE_UP);
    event port computeActionThread.outBrake -> outBrake in modes
               (RADAR_UP_CONSOLE_DOWN, RADAR_UP_CONSOLE_UP);
    event port computeActionThread.outThrottle -> outThrottle in modes
               (RADAR_UP_CONSOLE_UP, RADAR_DOWN_CONSOLE_UP);
    data port inSpeedometerData -> speedometerSampleThread.inData in modes
             (RADAR_UP_CONSOLE_UP, RADAR_UP_CONSOLE_DOWN, RADAR_DOWN_CONSOLE_UP);
    data port speedometerSampleThread.outActualSpeed ->
              computeActionThread.inActualSpeed in modes
              (RADAR_UP_CONSOLE_UP, RADAR_UP_CONSOLE_DOWN, RADAR_DOWN_CONSOLE_UP);
  modes
    RADAR_UP_CONSOLE_UP: initial mode;
    RADAR_UP_CONSOLE_DOWN : mode;
    RADAR_DOWN_CONSOLE_UP: mode;
    RADAR_DOWN_CONSOLE_DOWN: mode;
    RADAR_UP_CONSOLE_UP -[inRadarError]-> RADAR_DOWN_CONSOLE_UP;
    RADAR_UP_CONSOLE_DOWN -[inRadarError]-> RADAR_DOWN_CONSOLE_DOWN;
    RADAR_UP_CONSOLE_DOWN -[inConsoleToggle]-> RADAR_DOWN_CONSOLE_UP;
    RADAR_UP_CONSOLE_UP -[inConsoleToggle]-> RADAR_DOWN_CONSOLE_DOWN;
  properties
    Required_Connection => false applies to inRadarError in modes
                           (RADAR_DOWN_CONSOLE_UP, RADAR_DOWN_CONSOLE_DOWN);
    Required_Connection => false applies to inRadarData in modes
                           (RADAR_DOWN_CONSOLE_UP, RADAR_DOWN_CONSOLE_DOWN);
    Required_Connection => false applies to outBrake in modes
                           (RADAR_DOWN_CONSOLE_UP, RADAR_DOWN_CONSOLE_DOWN);
    Required_Connection => false applies to inConsoleToggle in modes
                           (RADAR_UP_CONSOLE_DOWN, RADAR_DOWN_CONSOLE_DOWN);
    Required_Connection => false applies to inConsoleData in modes
                           (RADAR_UP_CONSOLE_DOWN, RADAR_DOWN_CONSOLE_DOWN);
    Required_Connection => false applies to outThrottle in modes
                           (RADAR_UP_CONSOLE_DOWN, RADAR_DOWN_CONSOLE_DOWN);
    Required_Connection => false applies to inSpeedometerData in modes
                          (RADAR_DOWN_CONSOLE_DOWN);
end ControllerProcess.impl;

processor ControllerProcessor
  features
    memoryBus: requires bus access MemoryBus;
    deviceBus: requires bus access DeviceBus;
end ControllerProcessor;

processor implementation ControllerProcessor.impl
  properties
    Scheduling_Protocol => (RMS, EDF, Sporadicserver, SlackServer, ARINC653);
end ControllerProcessor.impl;

system ControllerSystem
  features
    inConsoleToggle: in event port;
    inConsoleData: in data port AccTypes::Stream;
    LanBus: requires bus access LanBus;
end ControllerSystem;

system implementation ControllerSystem.impl
  subcomponents
    computeProcess: process ControllerProcess.impl;
    computeProcessor: processor ControllerProcessor.impl;
    ramMemory: memory RamMemory;
    deviceBus: bus DeviceBus;
    memoryBus: bus memoryBus;
    radarDevice: device RadarDevice;
    speedomoterDevice: device SpeedometerDevice;
    wheelDevice: device SteeringWheelDevice;
    brakeDevice: device BrakeDevice;
    throttleDevice: device ThrottleDevice;
  connections
    bus access memoryBus -> computeProcessor.memoryBus;
    bus access memoryBus -> ramMemory.memoryBus;
    bus access deviceBus -> computeProcessor.deviceBus;
    bus access deviceBus -> radarDevice.deviceBus;
    bus access deviceBus -> speedomoterDevice.deviceBus;
    bus access deviceBus -> wheelDevice.deviceBus;
    bus access deviceBus -> brakeDevice.deviceBus;
    bus access deviceBus -> throttleDevice.deviceBus;
    event port inConsoleToggle -> computeProcess.inConsoleToggle;
    data port inConsoleData -> computeProcess.inConsoleData;
    event port radarDevice.outError -> computeProcess.inRadarError
               {Actual_Connection_Binding => reference deviceBus;};
    data port radarDevice.outData -> computeProcess.inRadarData
              {Actual_Connection_Binding => reference deviceBus;};
    data port speedomoterDevice.outData ->
              computeProcess.inSpeedometerData
              {Actual_Connection_Binding => reference deviceBus;};
    data port wheelDevice.outData -> radarDevice.inData
              {Actual_Connection_Binding => reference deviceBus;};
    event port computeProcess.outBrake -> brakeDevice.inBrake;
    event port computeProcess.outThrottle -> throttleDevice.inThrottle;
end ControllerSystem.impl;

system AccSystem
end AccSystem;

system implementation AccSystem.impl
  subcomponents
    consoleSystem: system ConsoleSystem;
    computeSystem: system ControllerSystem;
    lanBus: bus LanBus;
  connections
    bus access consoleSystem.lanBus -> computeSystem.lanBus;
    event port consoleSystem.outToggle -> computeSystem.inConsoleToggle
               {Actual_Connection_Binding => reference lanBus;};
    data port consoleSystem.outData -> computeSystem.inConsoleData
              {Actual_Connection_Binding => reference lanBus;};
end AccSystem.impl;
```

References

[1] J. Arlow and I. Neustadt. *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design.* Addison-Wesley Professional, second edition edition, 2005.

[2] J. Bengtsson. Adaptive Cruise Control and Driver Modeling. Master's thesis, Department of Automatic Control Lund Institute of Technology, Lund, 2001.

[3] P. Feiler and B. Lewis. SAE architecture analysis and design language (AADL) annex volume 1. Technical Report AS5506/1, Society of Automobile Engineers, 2006.

[4] P. H. Feiler, D. P. Gluch, and J. J. Hudak. The Architecture Analysis and Design Language (AADL): An Introduction. Technical Report CMU/SEI-2006-TN-011, Society of Automotive Engineers, 2006.

[5] P. H. Feiler, D. P. Gluch, J. J. Hudak, and B. A. Lewis. Embedded Systems Architecture Analysis Using SAE AADL. Technical Report CMU/SEI-2004-TN-005, Society of Automotive Engineers, 2004.

[6] J. Hitz, J. Koziol, and A. Lam. Safety evaluation results from the field operational test of an intelligent cruise control DICCE system. Technical Report 2000011352, Society of Automotive Engineers, 2000.

[7] B. Lewis and P. Feiler. Technical Report AS5506, Society of Automobile Engineers, 2004.

[8] R. Miles and K. Hamilton. *Learning UML 2.0.* O'Reilly Media, 2006.

[9] D. Pilone and N. Pitman. *UML 2.0 in a Nutshell.* O'Reilly Media, second edition edition, 2005.

[10] S. Vestal and J. Krueger. Technical and historical overview of metah. Technical Report MN 55418-1006, Honeywell Technology Center, 2000.

[11] X. Zhang. Intelligent Driving - Promotheus Approaches to Longitudinal Traffic Flow Control. In *VNIS '91*, pages 999–1010, 1991.