Laboration 2

Dzana Hanic, dhc17002
Amer Surkovic, asc17003

Table of content

Laboration 2.1

In this system we recognized different stakeholders like: project manager, developers, testers and integrators, mainainters, analysts, system architect, and end user. We also considered their interest in architectural views such as decomposition view, uses view, concurrency view, implementation view, and work assignment view.

Their relation is shown in Table 1., where we marked with "*x*" if a stakeholder has any interest with a corresponding view. For this purpose, we followed example from [1].

Views / Stakeholders	**Decomposition view**	**Uses view**	**Concurrency view**	**Implementation view**	**Work assignment view**
Project manager	x	x			x
Developers	x	x	x	x	x
Testers and integrators		x	x	x	x
Maintainers	x	x	x	x	x
Analysts	x	x	x		x
System architect	x	x	x	x	x
End user					

Table 1. Stakeholders and views

Architectural Views:

- **Decomposition view** - A view which is referred as a relation between different modules where larger units are decomposed into smaller and less complex units.
- **Uses view** - This view shows how the software modules interact in order to supply a correct service.
- **Concurrency view** - A view which observes dynamic aspects of a system, and it can be useful for identifying data consistency issues, deadlocks in a system, problems with resource contention, etc.
- **Implementation view** - A view which shows how software elements are mapped to the file structure in the development, integration of a system, or configuration control environments [1].
- **Work assignment view** - A view which is used for assigning responsibilities like implementation and integration of existing modules to the most appropriate teams. It is based on the skill set of the team members.

Stakeholders:

- **Project manager** - Person responsible for planning, execution, managing the people, resources and the scope of the project.[1]
- **Developers** - People responsible for coding and implementation of the system software.
- **Testers and integration (DevOps)** - People responsible for testing of the developed software and integration of the product to the working environment based on the customers needs.
- **Maintainers** - People responsible for maintaining the system and the product. Maintenance of the system includes everything from documentation to test results, not just software code.
- **Analysts** - People responsible for translating gathered data into sensible information helpful for the use for business growth and development.
- **System architect** - Person responsible for defining and designing system architecture of a computerized system in order to fulfill certain requirements defined by the customer and/or other stakeholders[2].
- **End user** - user of the developed system.

Project manager has an interest in *decomposition, uses and work assignment* views. This is a person who cares about schedule, resources, business plan, and is able to plan work assignments for a team [4]. Since manager has an interest with creating assignments and schedule, he/she would have interest in a work assignment view. For creating different assignments, an overview of system and its interaction is needed, and therefore, this person is interested in decomposition and uses view along with the work assignment view. Project manager is not interested in detailed description of a system, so other views are not important to this person.

Developers are interested in all of the views from Table 1. as these are mainly focused on a software itself (uses, decomposition, concurrency, deployment). Work assignment view is important for them too since they are assigned to separate tasks and different teams.

Testers and integrators are interested in all mentioned views (*uses, concurrency, Implementation, work assignment*) except in a *decomposition view*. Just like developers, they need information about software itself with an emphasis on behaviour of the system's specifications and interfaces. Therefore, uses view is more relevant for them when compared to decomposition view. Work assignment view is important for them too since they are assigned to project tasks.

Maintainers are interested in all views, just like the developers. This is because they both need to take into account the same constraints of a system.

Analysts are interested in all mentioned views except in implementation view. This is because their main focus lies in meeting system's quality objectives such as performance availability, security, modifiability, etc. They have to have insights into architecture of a system so they are able to use important tools for measuring these quality objectives. Having insight into *decomposition,*

[1] https://www.techopedia.com/definition/677/project-manager-pm
[2] https://en.wikipedia.org/wiki/Systems_architect

uses, and *concurrency* views will assure that they collect all necessary information from a system. Since they are part of a team, they are interested in *work assignment* view too. On the other side, detailed information from implementation view is more suitable for developers and testers.

System architect has an interest with all views from Table 1. It is important for an architect to be aware of all the key parts of a system, and, to have an overview of all these views. This is because an architect is in charge of creating and changing system's parts and functionalities.

End User is not interested in an architecture and details of a system. Therefore, we did not include his/her interest with mentioned views. However, we believe that end users would have interest in a deployment view so they understand their interaction with the important system functionalities.

We have spent approximately 3 hours/person on this assignment.

Laboration 2.2

In this section we will present our solution for design and documentation of a combined decomposition/uses view of the architecture.

Primary Presentation

This section will include two diagrams we designed to show decomposition/uses structure. First, we will describe a class diagram which gives more detailed information of the system including relevant relations, elements, and methods. The second diagram has a more informal notation - a diagram showing all modules and relations between them.

Figure 1. depicts ***Decomposition/Uses structure*** of the system. For showing how system interacts, we decided to use a class diagram with relevant methods and fields.

Figure 2. depicts modules from the system. The larger system - a vehicle with its functions, has been decomposed into smaller units (modules) based on existing functionalities. These modules are the same as the classes from the Figure 1.

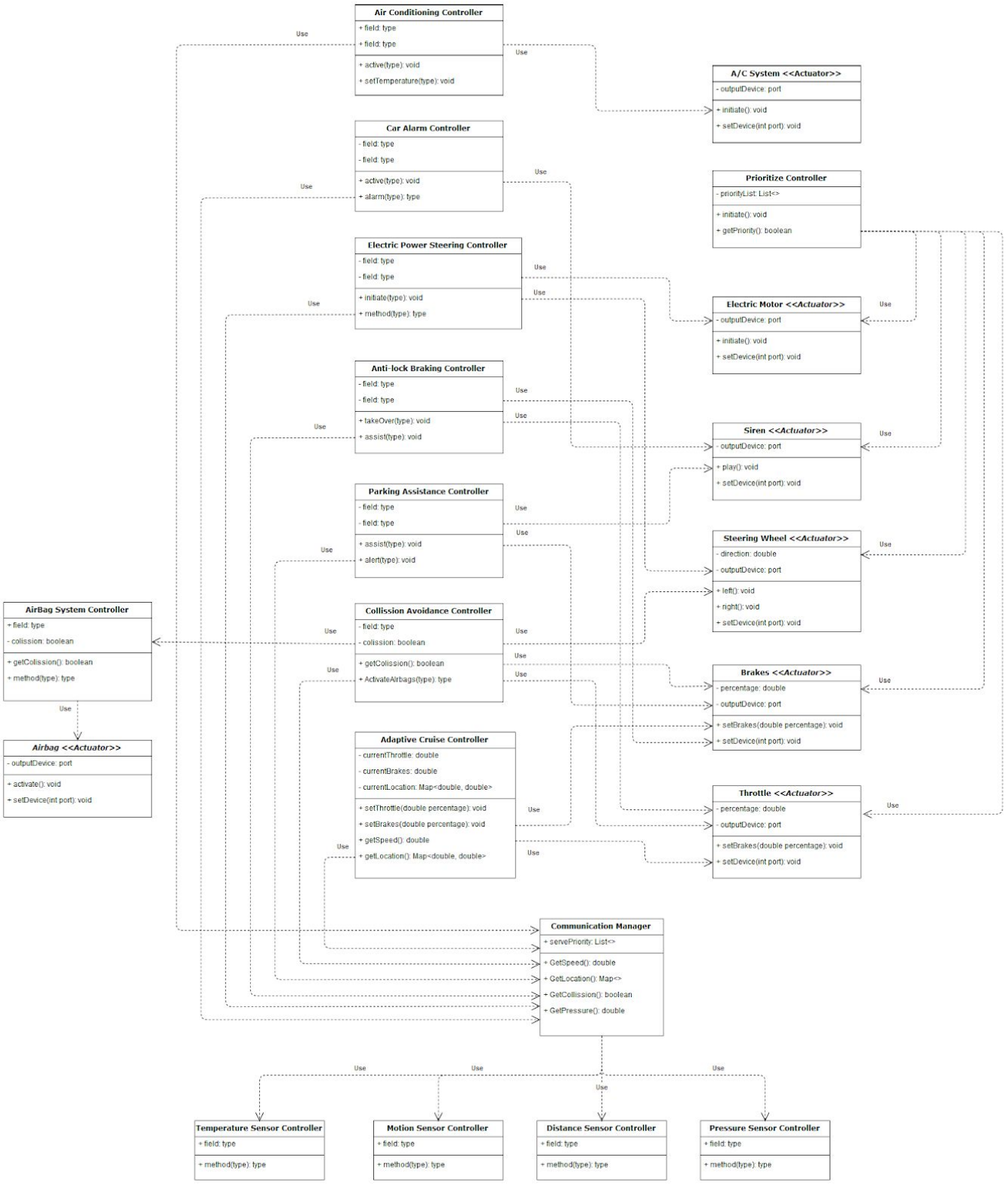

*Figure 1. Decomposition/Uses Structure**

**Diagram can be seen at the end of the document as well as on our Github page.*

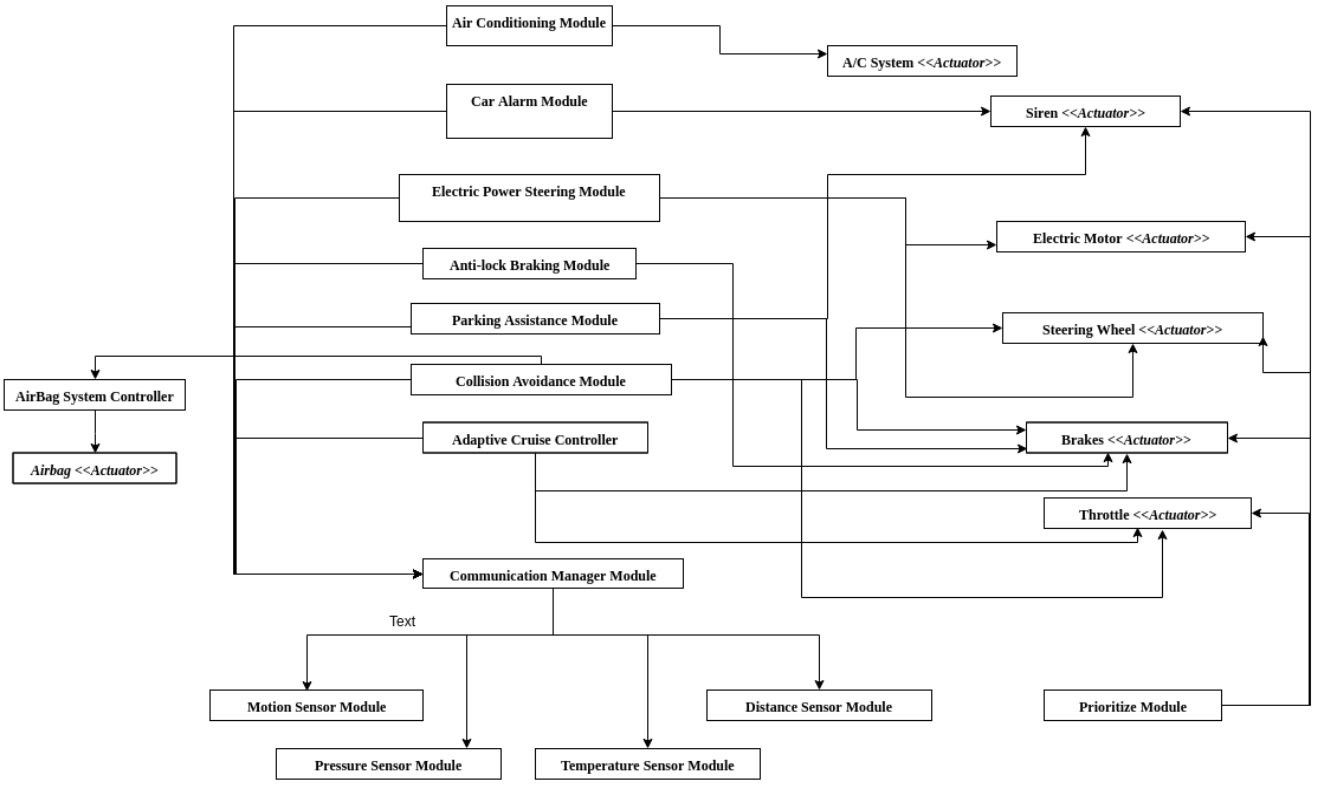

*Figure 2. System modules**

**Diagram can be seen at the end of the document as well as on our Github page.*

Element Catalogue

Elements and their properties

Elements of the system, seen in the decomposition/uses diagram, are as follows:

- **Adaptive Cruise Controller**
  This controller is responsible for getting information on car's speed and position by calling on the active sensors for the purpose of maintaining a desired speed and direction designated by the driver.
- **Collision Avoidance Controller**
  This controller is responsible for preventing any unwanted collision of the vehicle on the road. It is getting information on car's speed and position with respect to the other vehicles on the road including possible obstacles. If the dangerous situation presents itself, evade procedure is called and commands toward steering wheel, brakes and throttle are issued in order to avoid the collision.
- **Parking Assistance Controller**
  This controller is responsible for controlling the park assistance functionality of the vehicle. Controller gets the information from the distance sensor on any possible obstacle and if the car is too close to it, initiates the alarm to the driver.

- **Anti-lock Braking Controller**
  This controller is responsible for observing the break system of the vehicle. If the driver brakes, and holds the break for a certain amount of time, while the vehicle is at high speed - break lock may happen. In order to prevent this, anti-lock braking controller may initiate take over from the driver over the breaks and throttle to prevent locking from happening and bringing a vehicle to a safe stop.
- **Electric Power Steering Controller**
  This controller is responsible for issuing commands towards electric motor when the vehicle is at low speed to help the driver steer the vehicle. It uses steering wheel and electric motor components.
- **Car Alarm Controller**
  This controller takes information from the pressure sensor through communication manager and initiates the alarm if certain degree of pressure is met.
- **Prioritize Controller**
  This controller is of the utmost importance for it has a function of delegating which command from other controllers has priority over the actuators. This means that, for example, the instruction from Collision Avoidance Controller has priority above all others and if there exists the conflict over the resource of actuator, prioritize controller will grant collision avoidance controller priority.
- **Communication Manager**
  This controller is responsible for granting access and delegating which controller has access to which sensor in the system.

In addition to these, we selected the following two functionalities:

- **Air Conditioning Controller**
  This controller is responsible for reading the temperature from the temperature sensor and regulating it by using A/C system actuator. This controller is not being used by Prioritize Controller for there is no resources in conflict with other controllers.
- **AirBag System Controller**
  This is controller used specifically by Collision Avoidance Controller. When the collision avoidance controller fails to maneuver the vehicle away from the danger and the collision itself is detected, AirBag System Controller is called for as a last step measure for preventing harm to the vehicle occupants.

- **Sensor Controllers:**
  - **Motion Sensor Controller**
    Controller responsible for sending data from the motion sensor to the communication manager when asked for.
  - **Distance Sensor Controller**
    Controller responsible for sending data from the distance sensor to the communication manager when asked for.
  - **Pressure Sensor Controller**
    Controller responsible for sending data from the pressure sensor to the communication manager when asked for.

    - **Temperature Sensor Controller**
      Controller responsible for sending data from the temperature sensor to the communication manager when asked for.
- **Classes for modelling the connection with physical devices:**
    - **Electric Motor**
    - **Siren**
    - **Steering Wheel**
    - **Brakes**
    - **Throttle**
    - **A/C System**
    - **Airbag**

Relations and their properties

- **Adaptive Cruise Controller** uses **Communication manager** in order to get information from **Motion Sensor Controller** and **Distance Sensor Controller**. It then uses Throttle and Brakes actuator modules issuing commands on what to do.
- **Collision Avoidance Controller** uses **Communication manager** in order to get information from **Motion Sensor Controller** and **Distance Sensor Controller.** It then uses Steering Wheel, Brakes and Throttle actuator modules issuing commands on what to do.
- **Parking Assistance Controller** uses **Communication manager** in order to get information from **Distance Sensor Controller.** It then uses Siren and Brakes actuator modules issuing commands on what to do.
- **Anti-lock Braking Controller** uses **Communication manager** in order to get information from **Motion Sensor Controller.** It then uses Brakes and Throttle actuator modules issuing commands on what to do.
- **Electric Power Steering Controller** uses **Communication manager** in order to get information from **Motion Sensor Controller.** It then uses Steering Wheel and Electric Motor actuator modules issuing commands on what to do.
- **Car Alarm Controller** uses **Communication manager** in order to get information from **Pressure Sensor Controller.** It then uses Siren actuator modules issuing commands on what to do.
- **Prioritize Controller** uses all of the actuator modules in order to decide which of the actuators has the priority.
- **Air Conditioning Controller** uses **Communication manager** in order to get information from **Temperature Sensor Controller**. It then uses A/C System actuator module issuing command on what to do.
- **Collision Avoidance Controller** uses **AirBag System Controller** to initiate last preventive measure of activating airbags in case collision with another vehicle or a road obstacle happens.

Context Diagram

A context diagram of this system is depicted in Figure 3. **Hardware** components of the system include sensors and actuators. Sensors collect information on motion, distance, pressure and temperature from its driving environment. **Vehicle System Software** (closely described in Element Catalogue and Figure 1.) calculates collected data and sends a command to an actuator which has an influence on vehicle's physical parts.

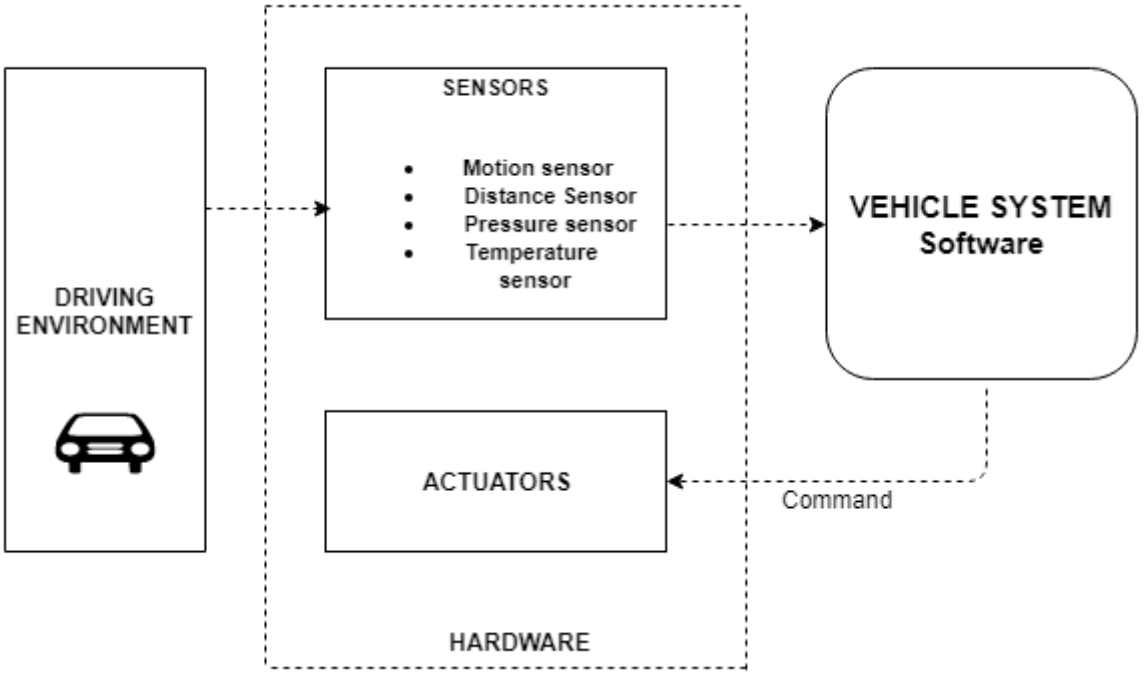

Figure 3. Context Diagram

Variability guide

In this section, we will show different variations of functionalities and hardware. A collection of different alternatives for an architecture has to be considered [3]. Including or excluding several hardware parts of this system directly affects on functionalities and architecture view of a system.

Firstly, we have discussed alternatives on having pressure and distance sensor.

- Pressure sensor
  Having a pressure sensor in a vehicle will allow an architect to include in his design Car Alarm functionalities. That implies including `getPressure()` function in `CommunicationManager`, developing a `CarAlarmController`, and having a communication with a siren.
  On the other side, if a vehicle does not support pressure sensor, Car Alarm functionalities could not be included in its architecture and implementation. Therefore, modules like `CarAlarmController` and `getPressure()` function in `CommunicationManager` could easily be removed from Figure 1.

- Distance sensor
  System collects and analyses data from distance sensor to develop functionalities like `Collision Avoidance`, `Parking Assistance`, `Adaptive Cruise Controller`. Without this sensor, those functionalities could not be included in architecture of the system.

In addition to this, we can also discuss different vehicle alternatives. For example, a vehicle with Collision Avoidance Functionality and Air-Bag System, and a vehicle without Air-Bag System.

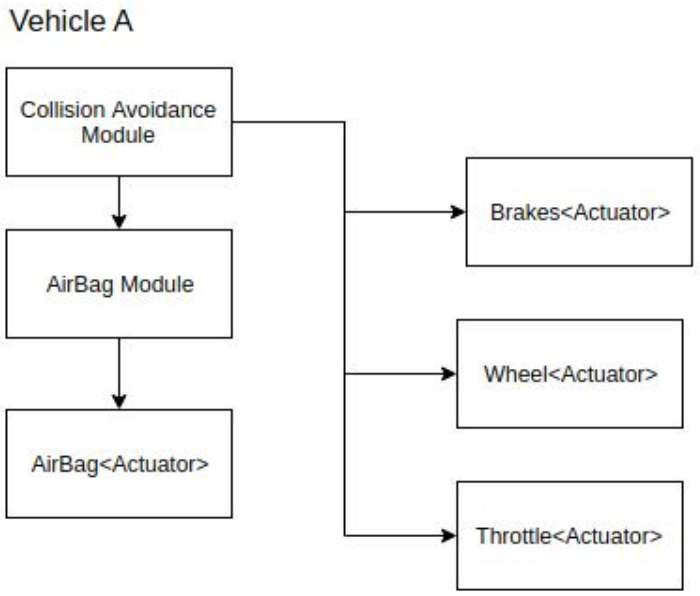

Figure 4. Vehicle with Collision Avoidance System and AirBag System

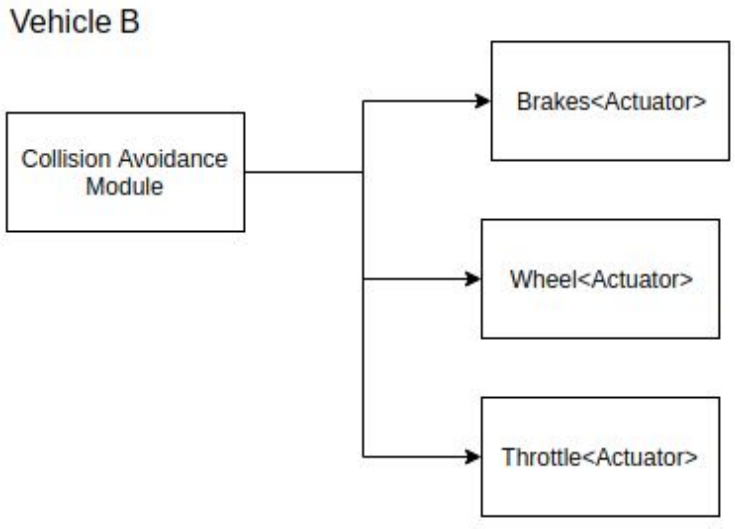

Figure 5. Vehicle without AirBag System

Figure 6. shows one of the notations for variability of modules followed from [3].

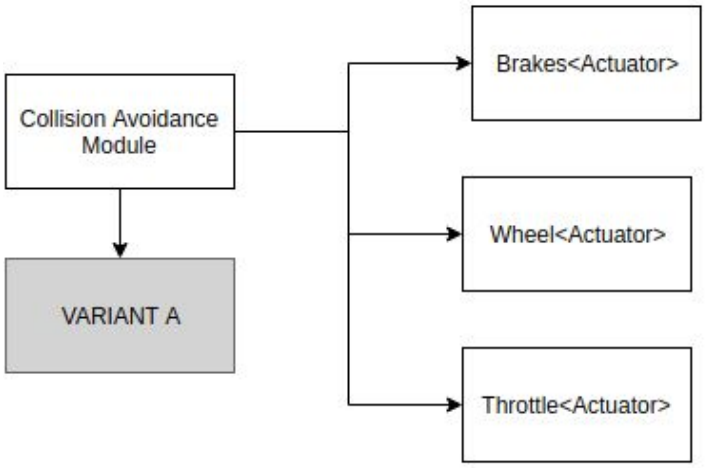

Figure 6. Module view with variations

Rationale

In our system design, we have applied decomposition up to the level where the optimal separation of concern has been done. This means that seperate parts of the system could be installed on the vehicles which may lack certain hardware components (sensors or other pieces of hardware) and still create a functional system. This is because all of the logical components of the system have been separated as not to interfere to the possible changes. We have followed **object oriented architecture style of design** where we abstracted logical units into separate classes and encapsulated internal parameters away from stakeholders who are not interested in the class implementation itself.

We have spent approximately 9 hours/person on this assignment.

Laboration 2.3

In this section, we will describe *concurrency view* of the system following the template from [1].

Primary Presentation

Graphical notation of a concurrency view is closely depicted in Figure 7.

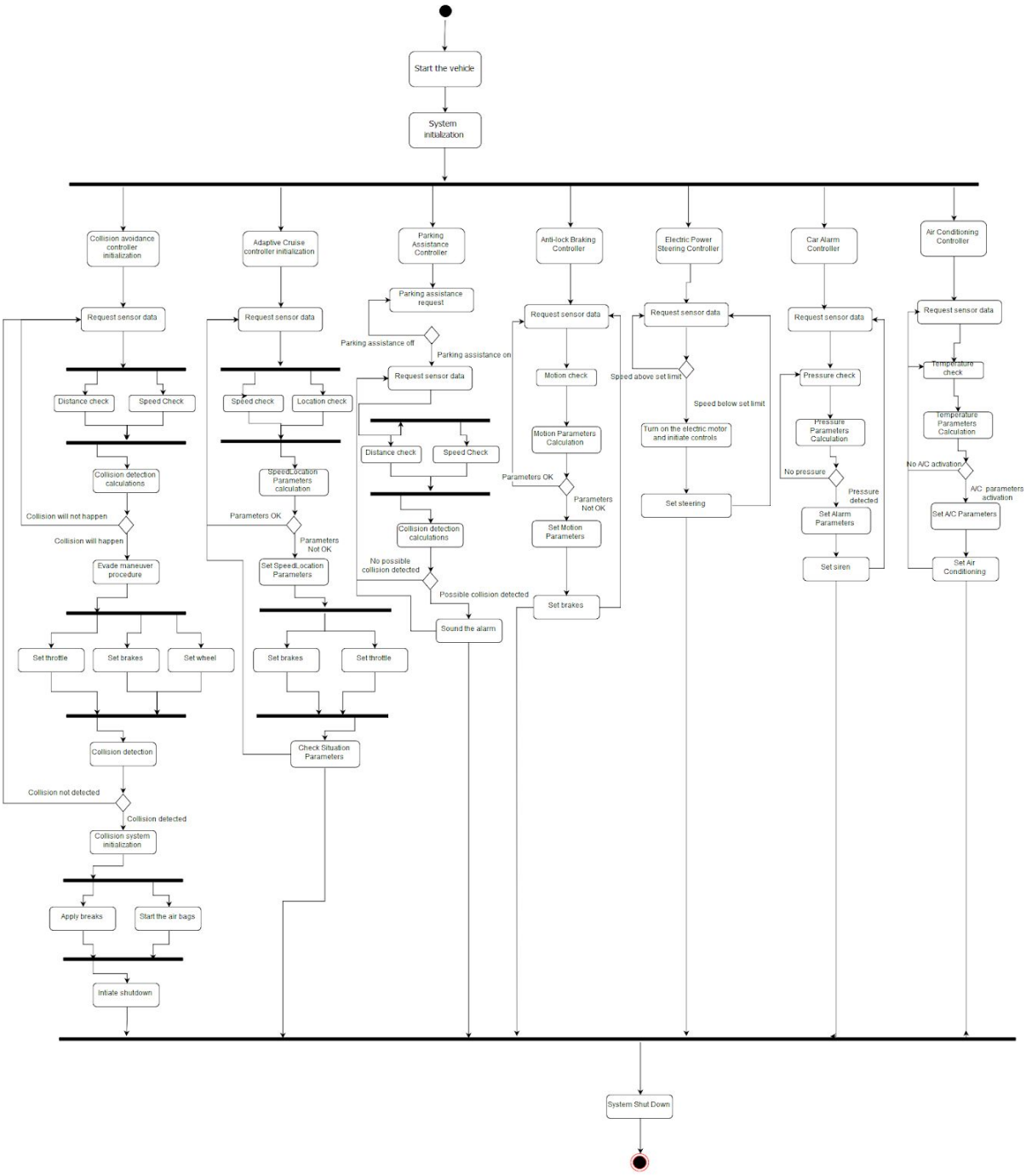

Figure 7. Concurrency View in Activity Diagram

Diagram can be seen at the end of the document as well as on our Github page.

Element Catalogue

Elements and their properties

Elements of our concurrency view graph are events of the system executed in certain period of time - when the system is active and running. Each event has its predecessor and successor meaning that we are able to see what happened before a certain event (what caused it) and what happens afterwards in time. This is important in order to understand the flow of data during execution time and to see how components interact to each other.
All subsystems explained in earlier part of this seminar are initialized when the vehicle is turned on and are working in parallel when the system is online.

Relations and their properties

The most important subsystem, running in parallel with other systems during software execution, is collision avoidance controller. It is responsible for avoiding collision on the road - and for calling an airbag system as a last resort in case collision happens. It has priority on all sensor data, through communication manager and priority on control over actuators. After initialized, CAC (collision avoidance controller) requests data from the sensors - on set periods of time. Using the collected data, it concludes if there is a possibility of collision. If not, it collects data until system is shutdown or until the conclusion is different. If the conclusion is that the collision will happen, collision avoidance procedure is called - where commands to wheel, steering and breaks actuators are issued. Controller is issuing commands until there is no more danger of collision - or until collision happens.If the collision happens, collision procedure is called applying brakes, activating airbags and bringing the vehicle to the full stop.

Second subsystem which runs in parallel with the others is Adaptive Cruise Control. System is initialized when activated by the driver. After that, it requests data from the sensors calculating desired speed and location. If the parameters are ok, controller will just repeat the step until system shutdown or until the parameters are not correct. If the parameters are not as desired, commands are sent to the break and throttle actuators to achieve desired state. Parameters are checked again where the steps are repeated until correct state is achieved.

Third subsystem is parking assistance controller. When on, it requests data from the sensors about speed of the vehicle and close obstacles on the road. If the possible collision is detected, alarm is sounded to the driver. Otherwise, the data is requested and decision is calculated until system is turned off.

Fourth subsystem is anti-lock braking controller which has a similar priority like the collision avoidance controller. It requests the data from the sensors, checks the motion of the car and if the extreme breaking is sensored, controlled braking is initiated.

Fifth subsystem is Electric Power Steering Controller which activates the electric motor if the speed of the car is below set limit. While the motor is on, steering is set by the controller. System works until the speed of the vehicle is below desired one.
Sixth subsystem which works only when vehicle is off is car alarm controller. It initiates alarm when the pressure on the vehicle doors or glass is registered.

Final subsystem running in parallel with other subsystems is air conditioning controller. It requests data on inside vehicle temperature and if the temperature is above desired one, air conditioning is activated.

Runtime Behaviour

Runtime behaviour of system elements is as follows:

Car Alarm Controller is active only when vehicle is locked meaning that it does not work in parallel with other controllers.
Electric Power Steering Controller works in parallel with other active controllers when it is activated - and that is when vehicle is moving under the designated speed. It has lower execution priority than collision avoidance controller and anti-lock braking controller.
Anti-lock Braking Controller works in parallel with other controllers during execution of software - from startup to the shutdown. It is a fail-safe mechanism and has the highest execution priority granted by Prioritize Controller.
Parking Assistance Controller is activated when parking. It works in parallel with collision avoidance controller and electric power steering controller. It. however is not active when actual vehicle is driving.
Collision Avoidance Controller is active during the whole system lifetime. It has highest priority together with anti-lock braking controller and is also a fail-safe mechanism in case of danger.
Adaptive Cruise Controller is activated by the driver and works in parallel with fail-safe mechanism controllers for better driving experience.
Communication Manager and Prioritize Controller work for the whole lifetime of software execution and are responsible for controlling data flow from controllers to actuators and sensors.
Actuator Controllers run the whole time and are responsible for sending execution commands to the physical actuators.
Sensor Controllers run the whole time and are responsible for getting sensory information from the sensors.

Variability guide

In this section, we will show different variations of a vehicle when observing Collision Avoidance functionality and how it can affect vehicle’s concurrency view.

Product A is a vehicle which has both Collision Avoidance functionality with Air-Bag System to react on vehicle’s crashes. A system will read and calculate data. When it detects collision, it will

react on a vehicle's throttle, brakes or a wheel. Then, it will continue working with sensor's data and react if a vehicle crashes. In this scenario, it will activate Air-Bag System.
Product B is a vehicle which doesn't have an option to detect collision, but it has Air-Bag System functionality. In this case, a system will only react if a car crashes by activating Air-Bag System.
Product C is a vehicle which doesn't have Air-Bag System, but it has Collision Avoidance functionalities.

After including or excluding several properties from a system, system's flow will be changed. Example of this change is shown in Figure 5 where different functions will be executed depending on type of a product..

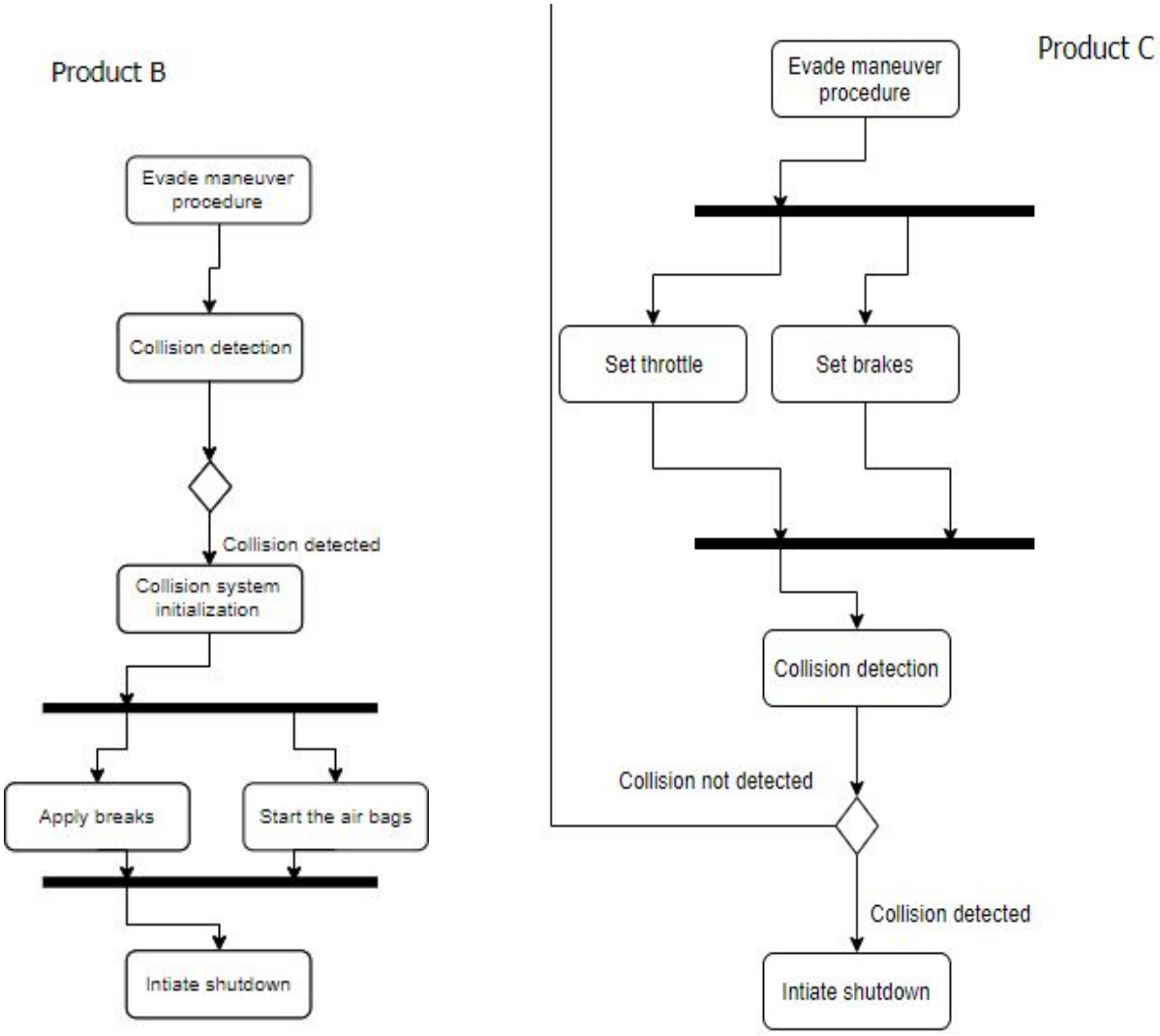

Figure 8. Variations of a product B and C

Rationale

Concurrency is a property of the system to perform several functions at the same time - where those functions either share resources or are working on their separate workspaces. In

non-concurrent systems, functions have to be executed sequentially meaning that next step in process can start when the last one has ended. Our system is concurrent and multiple modules perform a parallel. In order to show concurrent execution of our software we have decided to use the activity diagram. This is because in activity diagrams, concurrency can be shown implicitly or explicitly. With the implicit split and implicit join, we were able to represent what happens, and when, in time and which activities - modules - execute in parallel and how they behave for a certain external influence. Activity diagram represents a control flow which we found suitable to represent how our system works from the start up until the end of execution.

We have spent approximately 9 hours/person on this assignment.

References

[1] Len Bass, Paul Clements, Rick Kazman, *Software Architecture in Practice,* Third Edition
[2] “9.4 Documenting a View.” *9.4 Documenting a View :: Chapter 9. Documenting Software Architectures :: Part Two: Creating an Architecture :: Software architecture in practice :: Programming :: eTutorials.Org*, etutorials.org/Programming/Software architecture in practice, second edition/Part Two Creating an Architecture/Chapter 9. Documenting Software Architectures/9.4 Documenting a View/.
[3] Bachmann, Felix, and Len Bass. “Managing variability in software architectures.” Proceedings of the 2001 symposium on Software reusability putting software reuse in context - SSR 01, 2001, doi:10.1145/375212.375274.
[4] “Informit.” Stakeholders and Their Documentation Needs | InformIT, www.informit.com/articles/article.aspx?p=30695

Air Conditioning Module

A/C System <<*Actuator*>>

Car Alarm Module

Siren <<*Actuator*>>

Electric Power Steering Module

Electric Motor <<*Actuator*>>

Anti-lock Braking Module

Parking Assistance Module

Steering Wheel <<*Actuator*>>

Collision Avoidance Module

AirBag System Controller

Airbag <<Actuator>>

Adaptive Cruise Controller

Brakes <<*Actuator*>>

Throttle <<*Actuator*>>

Communication Manager Module

Text

Motion Sensor Module

Distance Sensor Module

Prioritize Module

Pressure Sensor Module

Temperature Sensor Module

Air Conditioning Controller
+ field: type
+ field: type
+ active(type): void
+ setTemperature(type): void

A/C System <<Actuator>>
- outputDevice: port
+ initiate(): void
+ setDevice(int port): void

Car Alarm Controller
- field: type
- field: type
+ active(type): void
+ alarm(type): type

Prioritize Controller
- priorityList: List<>
+ initiate(): void
+ getPriority(): boolean

Electric Power Steering Controller
- field: type
- field: type
+ initiate(type): void
+ method(type): type

Electric Motor <<*Actuator*>>
- outputDevice: port
+ initiate(): void
+ setDevice(int port): void

Anti-lock Braking Controller
- field: type
- field: type
+ takeOver(type): void
+ assist(type): void

Siren <<*Actuator*>>
- outputDevice: port
+ play(): void
+ setDevice(int port): void

Parking Assistance Controller
- field: type
- field: type
+ assist(type): void
+ alert(type): void

Steering Wheel <<*Actuator*>>
- direction: double
- outputDevice: port
+ left(): void
+ right(): void
+ setDevice(int port): void

AirBag System Controller
+ field: type
- colission: boolean
+ getColission(): boolean
+ method(type): type

Collission Avoidance Controller
- field: type
- colission: boolean
+ getColission(): boolean
+ ActivateAIrbags(type): type

Brakes <<*Actuator*>>
- percentage: double
- outputDevice: port
+ setBrakes(double percentage): void
+ setDevice(int port): void

***Airbag* <<*Actuator*>>**
- outputDevice: port
+ activate(): void
+ setDevice(int port): void

Adaptive Cruise Controller
- currentThrottle: double
- currentBrakes: double
- currentLocation: Map<double, double>
+ setThrottle(double percentage): void
+ setBrakes(double percentage): void
+ getSpeed(): double
+ getLocation(): Map<double, double>

Throttle <<*Actuator*>>
- percentage: double
- outputDevice: port
+ setBrakes(double percentage): void
+ setDevice(int port): void

Communication Manager
+ servePriority: List<>
+ GetSpeed(): double
+ GetLocation(): Map<>
+ GetCollission(): boolean
+ GetPressure(): double
+ GetTemperature(): double

Temperature Sensor Controller
+ field: type
+ method(type): type

Motion Sensor Controller
+ field: type
+ method(type): type

Distance Sensor Controller
+ field: type
+ method(type): type

Pressure Sensor Controller
+ field: type
+ method(type): type

Use

Start the vehicle

System initialization

Collision avoidance controller initialization

Request sensor data

Distance check

Speed Check

Collision detection calculations

Collision will not happen

Collision will happen

Evade maneuver procedure

Set throttle

Set brakes

Set wheel

Collision detection

Collision not detected

Collision detected

Collision system initialization

Apply breaks

Start the air bags

Intiate shutdown

Adaptive Cruise controller initialization

Request sensor data

Speed check

Location check

SpeedLocation Parameters calculation

Parameters OK

Parameters Not OK

Set SpeedLocation Parameters

Set brakes

Set throttle

Check Situation Parameters

Parking Assistance Controller

Parking assistance request

Parking assistance off

Parking assistance on

Request sensor data

Distance check

Speed Check

Collision detection calculations

No possible collision detected

Possible collision detected

Sound the alarm

Anti-lock Braking Controller

Request sensor data

Motion check

Motion Parameters Calculation

Parameters OK

Parameters Not OK

Set Motion Parameters

Set brakes

Electric Power Steering Controller

Request sensor data

Speed above set limit

Speed below set limit

Turn on the electric motor and initiate controls

Set steering

Car Alarm Controller

Request sensor data

Pressure check

Pressure Parameters Calculation

No pressure

Pressure detected

Set Alarm Parameters

Set siren

Air Conditioning Controller

Request sensor data

Temperature check

Temperature Parameters Calculation

No A/C activation

A/C parameters activation

Set A/C Parameters

Set Air Conditioning

System Shut Down