Module Guide for Mechatronics

Team 28, Controls Freaks
Abhishek Magdum
Dharak Verma
Jason Surendran
Laura Yang
Derek Paylor

January 19, 2023

1 Revision History

Date	Version	Notes
January 18, 2023	1.0	First Version

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Mechatronics	Explanation of program name
UC	Unlikely Change

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (**?**). We advocate a decomposition based on the principle of information hiding (**?**). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by **?**, as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (**?**). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adopted here is called design for change. The only anticipated change would be changes to the torque vectoring and traction control logic, that lives inside of the vehicle dynamics block.

AC1: Torque Vectoring

AC2: Traction Control

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Input/Output devices (Input: Pedals, steering sensors, motor and contactor feedback, Output: Speaker, motor commands).

UC2: FSAE rules and regulations

UC3: Module heirarchy

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Driver Interface

M2: Vehicle Dynamics

M3: Motor Interface

M4: Battery Monitor

M5: Governor

M6: Plant Model

Level 1	**Level 2**
Hardware-Hiding Module	Plant Model
Behaviour-Hiding Module	Driver Interface
	Vehicle Dynamics
	Motor Interface
	Battery Monitor
Software Decision Module	Governor

Table 1: Module Hierarchy

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 2.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by **?**. The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *Mechatronics* means the module will be implemented by the Mechatronics software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 Hardware Hiding Modules

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

7.1.1 Plant Module (M6)

Secrets: Drive input signals, HV and LV contactors.

Services: This module simulates the physical input signals to the various other models in order to test various functionality and edge cases.

Implemented By: driver_plant.slx

Type of Module: Simulink Model

7.2 Behaviour-Hiding Module

7.2.1 Driver Interface Module (M1)

Secrets:

Services: Pedal mapping

Implemented By: driver_interface_lib.slx

Type of Module: Simulink Model

7.2.2 Vehicle Dynamics Module (M2)

Secrets: The format and structure of the input data. ****check this

Services: Converts the input data into the data structure used by the input parameters module.

Implemented By: vehicle_dynamics_lib.slx

Type of Module: Simulink Model

7.2.3 Motor Interface Module (M3)

Secrets:

Services: Controls the left and right motors input parameters module.

Implemented By: motor_interface_lib.slx

Type of Module: Simulink Model

7.2.4 Battery Monitor Module (M4)

Secrets: The format and structure of the input data. ****check this

Services: Takes in contactor statuses for the battery and outputs battery status to the governor.

Implemented By: battery_monitor_lib.slx

Type of Module: Simulink Model

7.2.5 Governor Module (M5)

Secrets: The format and structure of the input data. ****check this

Services: Monitors inputs from Driver Interface, Vehicle Dynamics, Motor Interface and Battery Monitor modules and sends commands to these modules based on the current state of the vehicle.

Implemented By: governor_lib.slx

Type of Module: Simulink Model

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
NFR1, NFR2, NFR3, NFR4, NFR5, NFR6	M1
VDR1, VDR2, NFR1, NFR2, NFR3, NFR5, NFR6	M2
TMR1, TMR2, TMR2, TMR4, TMR5, TMR6, TMR7, NFR1, NFR2, NFR3, NFR4, NFR5, NFR6	M3
AMR1, AMR2, AMR3, AMR4, AMR5, AMR6, AMR7, AMR8, AMR9, NFR1, NFR2, NFR3, NFR5, NFR6	M4
MSR1, MSR2, MSR3, MSR4, MSR5, MSR6, MSR7, MSR8, MSR9, NFR1, NFR2, NFR3, NFR5, NFR6	M5

Table 2: Trace Between Requirements and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

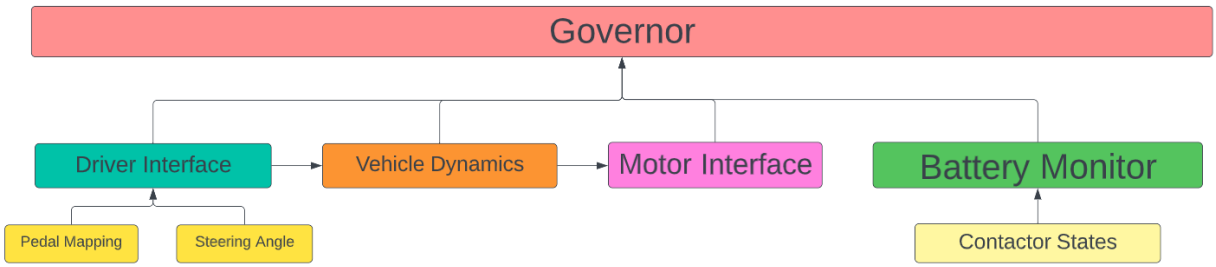

Figure 1: Use hierarchy among modules