Module Guide for SmartLock 4TB6 - Mechatronics Capstone

Team #5, Locked & Loaded
Abi Nevo, nevoa
Elsa Bassi, bassie
Steffi Ralph, ralphs1
Abdul Iqbal, iqbala18
Stephen De Jong, dejons1
Anthony Shenouda, shenoa2

April 7, 2023

Table 1: Revision History

Date	Developer	Changes
02-01-23	Elsa	Likely and Unlikely Changes
10-01-23	Elsa	Modules
15-01-23	Elsa	Remaining Contents
16-01-23	Elsa	References
02-03-23	Abi	Rev 1 Revisions–Modules reworked

1 Reference Material

This section records information for easy reference.

1.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SRS	Software Requirements Specification
4TB6 - Mechatronics Capstone	Explanation of program name
UC	Unlikely Change

Contents

List of Tables

List of Figures

2 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 (Parnas et al, 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972a). 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 Parnas et Al (1984), 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, (Parnas et Al, 1984). 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 3 lists the anticipated and unlikely changes of the software requirements. Section 4 summarizes the module decomposition that was constructed according to the likely changes. Section 5 specifies the connections between the software requirements and the modules. Section 6 gives a detailed description of the modules. Section 7 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 8 describes the use relation between modules.

3 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 3.1, and unlikely changes are listed in Section 3.2.

3.1 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 adapted here is called design for change.

AC1: The format of the initial input data and parameters on the App.

AC2: The format of the output data on the App.

AC3: The implementation of the transmission of the 'disengage' signal.

AC4: How the battery status calculation is defined within the App.

AC5: How the Arduino allows/disallows power to the load (solenoid) I.e., the implementation of the switch (transistor).

AC6: The implementation of the location determinant feature.

AC7: The size of the battery.

AC8: The size of the solenoid.

AC9: The implementation of the locking mechanism.

AC10: The implementation of the lock frame.

3.2 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: The I/O devices (iPhone or android smartphone).

UC2: Use of the Arduino Nano BLE board, and use of a transistor and solenoid.

UC3: Output data are displayed to the output device (App).

UC4: Output data can be verified by checking the lock frame.

UC5: The goal of the system (wirelessly and hands-free disengage the bike lock).

UC6: There will always be a source of input data external to the software.

4 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: User Input to Phone Module

M2: Solenoid Actuation Module

M3: Arduino Bluetooth Communication Module

M4: Mobile App Bluetooth Communication Module

M5: Battery Status Module

M6: User Disengage Module

M7: Hardware Disengage

M8: Location Module

M9: Battery Module

M10: Locking Mechanism Module

M11: Lock Frame Module

5 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 3.

Level 1	Level 2
Hardware-Hiding Module	User Input to Phone Module Solenoid Actuation Module
Behaviour-Hiding Module	Arduino Bluetooth Communication Module Mobile App Bluetooth Communication Module Battery Status Module Location Module Lock Frame Module
Software Decision Module	User Disengage Module Hardware Disengage Module Battery Module Locking Mechanism Module

Table 2: Module Hierarchy

6 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al, (1984). 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. *4TB6 - Mechatronics Capstone* means the module will be implemented by the 4TB6 - Mechatronics Capstone 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.

6.1 Hardware Hiding Modules

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

Services: Serves as 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 accept inputs.

Implemented By: –

6.1.1 User Input to Phone Module (M1)

Secrets: How the mobile app reacts to user's touch at any point.

Services: Allows user to interface with the mobile app via touch.

Implemented By: Phone OS

6.1.2 Solenoid Actuation Module (M2)

Secrets: Logic and implementation of the solenoid actuation depending on signal sent from Arduino.

Services: Actuates the solenoid when appropriate signal is sent from Arduino.

Implemented By: Circuit

6.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

6.2.1 Arduino Bluetooth Communication Module (M3)

Secrets: Creation of infrastructure for Bluetooth communication on Arduino, searches for central device to connect to, and can successfully connect to a central device.

Services: Allows delivery of Bluetooth signals from the central connected device.

Implemented By: Arduino

6.2.2 Mobile App Bluetooth Communication and User Disengage Module (M4)

Secrets: Creation of infrastructure for Bluetooth communication on the app, searches for peripheral devices to connect to, and can then successfully connect to a peripheral device. Also transmits disengage signal.

Services: Allows sending of Bluetooth signals to the connected peripheral device. Then, upon pressing "Disengage" button on app, app sends required signal to peripheral device.

Implemented By: App

6.2.3 Battery Status Module (M5)

Secrets: The determination of the current level of the battery to inform the user when replacement is required.

Services: The battery level is calculated within the App based on some inherent battery properties. It is then displayed on the GUI as 'Battery Status'.

Implemented By: App

6.2.4 Location Module (M8)

Secrets: The implementation of the location determinant.

Services: The algorithm that governs the geocaching of the bike's location upon locking.

Implemented By: App

6.2.5 Lock Frame Module (M11)

Secrets: The design of the physical 'lock frame' that connects the bike and the external frame it is being 'locked' to.

Services: Is the interface that connects the user, the App and the locking mechanism that allows the user to utilize the entire system.

Implemented By: Mechanical System

6.3 Electrical/Mechanical/Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

6.3.1 Hardware Disengage Module(M7)

Secrets: Logic of HIGH and LOW signal sent from Arduino.

Services: The implementation, logic and transmission of the power signal sent from the the Arduino to the transistor, which acts as a switch to give power to the solenoid.

Implemented By: Arduino

6.3.2 Battery Module (M9)

Secrets: The specifications of the circuit power supply.

Services: The type and size of the battery which supplies power to the Arduino and the electromagnet.

Implemented By: Electrical Circuit

6.3.3 Locking Mechanism Module (M10)

Secrets: The design of the moving part that engages and disengages the lock frame.

Services: Responds to the magnetic force supplied by the electromagnet in such a way that it engages and disengages the lock frame.

Implemented By: Mechanical System

7 Traceability Matrix

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

Req.	**Modules**
FR1	M1, M2, M3, M4, M10
FR2	M8
FR3	M10
FR4	M3, M4
FR5	M11
FR6	M9, M5
FR7	M8
FR8	M11

Table 3: Trace Between Requirements and Modules

AC	**Modules**
AC1	M1
AC2	M4
AC3	M7
AC4	M5
AC5	M7
AC6	M8
AC7	M9
AC8	M2
AC9	M10
AC10	M11

Table 4: Trace Between Anticipated Changes and Modules

8 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas, (1978), 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.

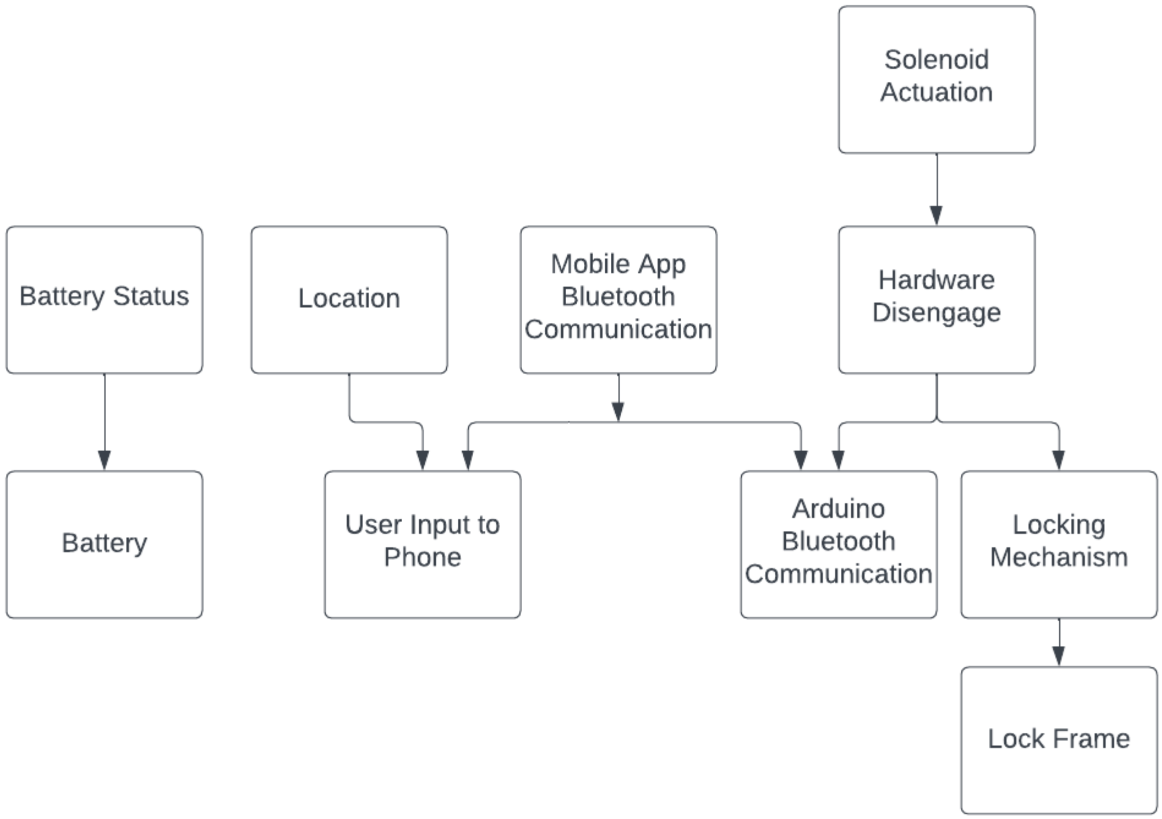

Figure 1: Use Hierarchy Among Modules