Module Guide for Radio Signal Strength Calculator

Xingzhi Liu

November 21, 2020

1 Revision History

Date	Version	Notes
Nov 19, 2020	1.0	First Draft

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
ProgName	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 (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). 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 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 adapted here is called design for change.

AC1: The specific hardware on which the software is running.

AC2: The overall control of the calculation.

AC3: The format of the input parameters.

AC4: The constraints on the input parameters.

AC5: The format of the output data.

AC6: What composes the received signal at the sampling point(e.g. direct signal, reflection, refraction, diffraction... What shall we consider in our calculation and what shall we neglect?).

AC7: How the radio propagation loss models with direct paths are defined using the input parameters.

AC8: How the intersections of line segments are detected.

AC9: How the linear segments of first-order reflected signals' paths are found.

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: File and/or Keyboard, Output: File, Memory, and/or Screen).

UC2: Walls are linear with finite lengths.

UC3: Only one transmitter works in the area of simulation.

UC4: Line segments are represented with a starting and an ending point, in addition to the linear equation (e.g. $m1 \cdot x + m2 \cdot y = k$) of that line.

UC5: A linear equation $m1 \cdot x + m2 \cdot y = k$ will be represented as a series of 3 parameters: $m1, m2$, and k in RSSC.

UC6: The constraints on the output data will not not change due to the law of conservation of energy.

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: Hardware-Hiding Module

M2: Input Parameters

M3: Output

M4: Control Module

M5: Received Signal

M6: Specification Parameters Module

M7: Point

M8: Wall

M9: Input Floor Map

M10: Equation Finder

M11: Linear Signal Path

M12: Intersection

M13: Linear Path Loss

M14: Line-Of-Sight Signal

M15: Specular Reflection

M16: First-Order Reflection

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding	Input Parameters
	Output
	Control Module
	Received Signal Strength
	Specification Parameters Module
Software Decision	Point
	Wall
	Floor Map
	Equation Finder
	Linear Signal Path
	Intersection
	Linear Path Loss
	Line-Of-Sight Signal
	Specular Reflection
	First Order Reflection Signal

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 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. *ProgName* means the module will be implemented by the ProgName 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 (M1)

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

7.2 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: –

7.2.1 Input Parameters Module (M2)

Secrets: The data structure for input parameters, how the values are input and verified.

Services: Gets input from user, verifies that the inputs meet the physical and software constraints, and stores input for other modules to read.

Implemented By: RSSC

7.2.2 Output Module (M3)

Secrets: The format and structure for output data.

Services: Outputs calculation results to a text file.

Implemented By: RSSC

7.2.3 Control Module (M4)

Secrets: The algorithm for coordinating the running of the program.

Services: Provide the main program.

Implemented By: RSSC

7.2.4 Received Signal Strength Module (M5)

Secrets: The solver for the received radio signal strength.

Services: Combines signals from different paths and find the total strength of the combined signal. Also verifies whether the combined signal satisfies the law of conservation of energy or not.

Implemented By: RSSC

7.2.5 Specification Parameters Module (M6)

Secrets: The values of the specification patameters.

Services: Read access for the specification parameters.

Implemented By: RSSC

7.3 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: –

7.3.1 Point Module (M7)

Secrets: The data structure for a Point type.

Services: Provides operations of creating points, setting point position and reading point position.

Implemented By: RSSC

7.3.2 Wall Module (M8)

Secrets: The data structure for a Wall type.

Services: Provides operations of creating a wall, as well as setting and reading of wall parameters including transmittance, reflectance, and starting or ending points of the wall.

Implemented By: RSSC

7.3.3 Floor Map Module (M9)

Secrets: The data structure for a Floor Map type.

Services: Provides operations of creating and reading of a Floor Map from the user input.

Implemented By: RSSC

7.3.4 Equation Finder Module (M10)

Secrets: The algorithm to find proper linear equation representation for a line segment.

Services: Find coefficients of a line segment's linear equation from the vertices of the segment.

Implemented By: RSSC

7.3.5 Linear Signal Path Module (M11)

Secrets: The data structure for a Linear Signal Path type.

Services: Provides operations of creating a Linear Signal Path, as well as reading of path parameters including starting and ending vertices, linear equation parameters and path length.

Implemented By: RSSC

7.3.6 Intersection Module (M12)

Secrets: The algorithm to find a signal path's intersection point on a wall.

Services: Finds the potential intersection point of the wall's line and the signal path's line and determines the validity of the intersection (whether the potential intersection point is both on the wall and on the signal path).

Implemented By: RSSC

7.3.7 Linear Path Loss Module (M13)

Secrets: The algorithm to find the propagation loss along a linear signal path.

Services: Calculate the percentage of power that a radio signal would remain after travelling through a given linear signal path.

Implemented By: RSSC

7.3.8 Line-Of-Sight Signal Module (M14)

Secrets: The data structure for a Line-Of-Sight Signal type.

Services: Provides operations of creating a Line-Of-Sight Signal and reading of parameters including the signal's strength and phase angle at its destination, and the length of its transmission path.

Implemented By: RSSC

7.3.9 Specular Reflection Module (M15)

Secrets: The algorithm to find linear segments of a reflected signal path.

Services: For a signal given an origin, a destination, and a potential reflection surface, determines the linear signal path from the origin to the reflection surface, the path from the reflection surface to the destination, and the validity of the whole path.

Implemented By: RSSC

7.3.10 First-Order Reflection Signal Module (M16)

Secrets: The data structure for a First-Order Reflection Signal type.

Services: Provides operations of creating a First-Order Reflection Signal, and reading of parameters including the signal's strength and phase angle at its destination.

Implemented By: RSSC

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes. Two of the anticipated changes (AC3 and AC4) are mapped to the same module (M**??**). The reason for this is that the services to store and verify user inputs in this module will always be provided together.

Req.	**Modules**
R1	M1, M2, M4
R2	M2, M6
R3	M5, M7, M8, M9, M10, M11, M12, M13, M14, M15, M16
R4	M5
R5	M3

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC1	M1
AC2	M4
AC3	M2
AC4	M2
AC5	M3
AC6	M5
AC7	M13
AC8	M12
AC9	M15

Table 3: Trace Between Anticipated Changes and Modules

9 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.

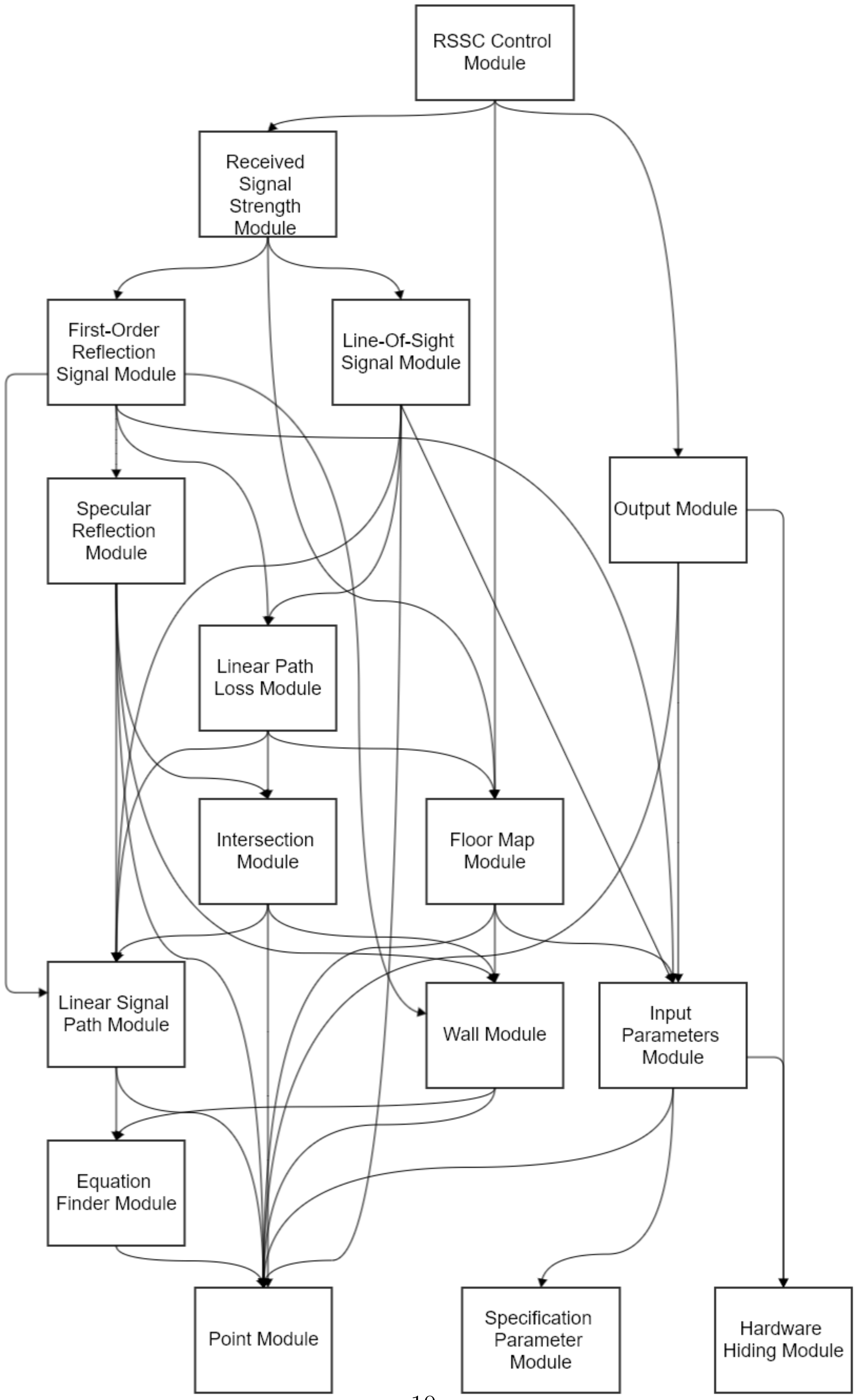

Figure 1: Use hierarchy among modules

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.