

Module Guide for IP simulator

Mina Mahdipour

March 2, 2023

1 Revision History

Date	Version	Notes
March 1, 2023	1.0	Created the document and fill initial data
Date 2	1.1	Notes

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
IP	Inverted Pendulum
M	Module
MG	Module Guide
ODE	Ordinary Differential Equation
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
UC	Unlikely Change

Contents

1	Revision History	i
2	Reference Material	ii
2.1	Abbreviations and Acronyms	ii
3	Introduction	1
4	Anticipated and Unlikely Changes	2
4.1	Anticipated Changes	2
4.2	Unlikely Changes	2
5	Module Hierarchy	3
6	Connection Between Requirements and Design	4
7	Module Decomposition	4
7.1	Hardware Hiding Modules (M1)	4
7.2	Behaviour-Hiding Module	4
7.2.1	Input Parameters Module (M2)	5
7.2.2	Output Parameters Module (M3)	5
7.3	Software Decision Module	5
7.3.1	ODE Solver Module (M4)	6
7.3.2	Plotting Module (M5)	6
8	Traceability Matrix	6
9	Use Hierarchy Between Modules	6

List of Tables

1	Module Hierarchy	3
2	Trace Between Requirements and Modules	4
3	Trace Between Anticipated Changes and Modules	6

List of Figures

1	Use hierarchy among modules	7
---	---------------------------------------	---

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 [3]. We advocate a decomposition based on the principle of information hiding [1]. 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 [3], 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 [3]. 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. Anticipated changes are numbered by **AC** followed by a number.

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

AC2: The format of the initial input data.

AC3: The format of the input parameters.

AC4: The constraints on the input parameters.

AC5: The format of the final output data.

AC6: The choice of data structures used for storing and manipulating the data.

AC7: The constraints on the output results.

AC8: How the governing Ordinary Differential Equations (ODEs) are defined using the input parameters.

AC9: The algorithm used for solving the equations of motions ODEs.

AC10: The implementation of plotting the output.

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

- UC3:** Output data are displayed to the output device.
- UC4:** The goal of the system is to calculate the position of cart and the angle of pendulum.
- UC5:** The ODEs for the cart and pendulum motions can be defined using parameters defined in the input parameters module.
- UC6:** How the equations of motions of the cart and the pendulum are defined using the specification of the system.

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. Modules are numbered by **M** followed by a number.

M1: Hardware-Hiding Module

M2: Input Parameters Module

M3: Output Parameters Module

M4: ODE Solver Module

M5: Plotting Module

Note that **M1** is a commonly used module and is already implemented by the operating system. It will not be reimplemented. Similarly, **M??**, **M4** and **M5** are already available in Python and will not be reimplemented.

Level 1	Level 2
Hardware-Hiding Module	
	Input Parameters Module
Behaviour-Hiding Module	Output Parameters Module
	ODE Solver Module
Software Decision Module	Plotting Module

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](#).

Req.	Modules
R1	M1, M2
R2	M2
R3	M4
R4	M1, M3, M5

Table 2: Trace Between Requirements and Modules

7 Module Decomposition

Modules are decomposed according to the principle of “information hiding” proposed by Parnas et al. [3]. 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. *IP Simulator* means the module will be implemented by the IP Simulator 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 how the values are verified. The load and verify secrets are isolated to their own access programs (like submodules). This, combined with the fact that all of the services are invoked together, suggests that the one module one secret rule can be relaxed here.

Services: Gets input from user (including material properties, processing conditions, and numerical parameters), stores input and verifies that the input parameters comply with physical and software constraints. Throws an error if a parameter violates a physical constraint. Throws a warning if a parameter violates a software constraint. Stored parameters can be read individually, but write access is only to redefine the entire set of inputs.

Implemented By: IP Simulator

7.2.2 Output Parameters Module (M3)

Secrets: The format and structure of the output data.

Services: Outputs the results of the calculations, including the position of the cart and the angle of the pendulum, stores them and verifies that the output parameters comply with physical and software constraints.

Implemented By: IP Simulator

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 ODE Solver Module (M4)

Secrets: The algorithm to solve a system of first or higher orders ODEs initial value problem from a given starting time until the given event function shows termination.

Services: Solves an ODE using the governing equation, initial conditions, event function and numerical parameters.

Implemented By: Python

7.3.2 Plotting Module (M5)

Secrets: The data structures and algorithms for plotting data graphically.

Services: Provides a plot function.

Implemented By: Python

8 Traceability Matrix

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

AC	Modules
AC1	M1
AC2	
AC3	
AC4	
AC5	M3
AC6	
AC7	M3
AC8	M4
AC9	M4
AC10	M5

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. [2] 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 tree. 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

- [1] David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.
- [2] 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.
- [3] 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.