CAS 741: Module Guide

Dynamical Systems: MPS

Karol Serkis
serkiskj@mcmaster.ca
GitHub: karolserkis

December 14, 2018

1 Revision History

Date	Version	Notes
October 31, 2018	1.0	First full draft for submission

Contents

List of Tables

List of Figures

2 Introduction

The module guide document is providing to the reader the decomposition of MPS into smaller pieces to help the implementation phase. 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 laid 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 used 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 specific hardware on which MPS software is running.

AC2: The format of the initial input data.

AC3: The 3D coordinate space to a 2D coordinate space.

AC4: The format of the Lagrangian/Hamiltonian (2D vs 3D).

AC5: The (linear) solver algorithm.

AC6: The content of the output (eg. 2D Poincare plot instead of 3D).

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: 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. [You can remove this UC. I know it was in the template, but it doesn't make sense to me. I've removed it from the template. —SS]

UC3: The goal statements of MPS (see SRS; extensions to SRS are still possible, but current goals should not change)

UC4: The data types (real number) interface device with user [This looks like two unlikely changes in one. The physics of the problem is for real numbers, so I'm not sure you need to say this. You already mention input devices in the first UC. —SS]

4 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 (described in MG_Hardware)

M2: MPS Control Module (described in MG_Control)

M3: MPS GUI Module (described MG_GUI)

M4: User Input Parameters Module (described in MG_InputFormat)

M5: Lagrangian Module (described in MG_LA)

M6: Hamiltonian Module (described in MG_HA)

M7: Data Structure Module (described in MG_DataStruct)

M8: Generic GUI/Plot Module (described in MG_GUIplot)

M9: Generic Trajectory Simulation GUI Module (described in MG_SIM)

Level 1	**Level 2**
Hardware-Hiding Module	
	MPS Control Module
	MPS GUI Module
	User Input Parameters Module
Behaviour-Hiding Module	Data Structure Module
	Generic Trajectory Simulation GUI Module
Software Decision Module	Generic GUI/Plot Module
	Lagrangian Module
	Hamiltonian Module

Table 1: Module Hierarchy

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 2.

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. Also indicate if the module will be implemented specifically for the 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. Whether or not this module is implemented depends on the programming language selected.

6.1 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

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 MPS Control Module (M2)

Secrets: Execution flow of MPS . [At the end of a sentence you want to just use progname without the parentheses. —SS]

Services: Calls the different modules in the appropriate order.

Implemented By: MPS

6.2.2 MPS GUI (M3)

Secrets: Methods to interact with user to make run MPS . [Are you combining your GUI and your control? You might want to consider the Model View Controller pattern. This would mean that the view wouldn't actually control anything. —SS]

Services: Serves as interface between the user and the software through the hardware by outputting calculation results and collecting user information (user inputs) for the calculation modules.

Implemented By: MPS

6.2.3 User Input Module (M4)

Secrets: The format and structure of the input data.

Services: Loads, verifies and stores the input data into the appropriate data and object structure.

Implemented By: MPS

6.3 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: –

6.3.1 Lagrangian Module (M5)

Secrets: Lagrangian calculation/algorithm.

Services: Calculates the variation of Lagrangian for each pendulum in the chain (each element of the array).

Implemented By: MPS

6.3.2 Hamiltonian Module (M6)

Secrets: Hamiltonian calculation/algorithm.

Services: Calculates the variation of Hamiltonian for each pendulum in the chain (each element of the array).

Implemented By: MPS

6.3.3 Data Structure Module (M7)

Secrets: Data format for an image.

Services: Provides convenient format to store, read and manipulate all elements (pixel) from an image.

Implemented By: Python library

6.3.4 Generic GUI/Plot Module (M8)

Secrets: Generic methods to interact with the user.

Services: Provides the generic interface methods such as buttons, windows, plotting, entry fields to interact with a user.

Implemented By: Python library

6.3.5 Trajectory Simulation Module (M9)

Secrets: Algorithm to simulate the plot trajectory of the chain of pendula.

Services: Simulates the chain of pendula in space using the Lagrangian and the Hamiltonian.

Implemented By: MPS

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**
R1	M2, M3, M4, M5, M6, M7, M8, M9
M??, M??	
R2	M2, M3, M4, M5, M6
R3	M4, M5, M6, M7, M8, M9
R4	M4, M5, M6, M7, M8, M9

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC1	M1, M9
AC2	M4, M5, M6, M7
AC3	M2, M3, M8, M9
AC4	M2, M3, M8, M9
AC5	M4, M5, M6, M7
AC6	M4, M5, M6, M7

Table 3: 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.

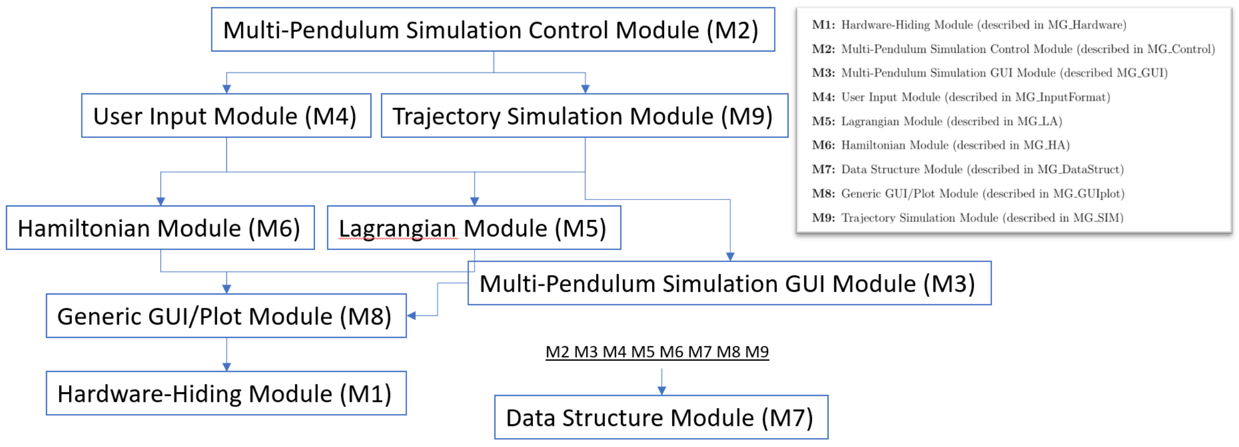
Multi-Pendulum Simulation Control Module (M2)
User Input Module (M4)
Trajectory Simulation Module (M9)
Hamiltonian Module (M6)
Lagrangian Module (M5)
Multi-Pendulum Simulation GUI Module (M3)
Generic GUI/Plot Module (M8)
Hardware-Hiding Module (M1)
M2 M3 M4 M5 M6 M7 M8 M9
Data Structure Module (M7)
M1: Hardware-Hiding Module (described in MG_Hardware)
M2: Multi-Pendulum Simulation Control Module (described in MG_Control)
M3: Multi-Pendulum Simulation GUI Module (described MG_GUI)
M4: User Input Module (described in MG_InputFormat)
M5: Lagrangian Module (described in MG_LA)
M6: Hamiltonian Module (described in MG_HA)
M7: Data Structure Module (described in MG_DataStruct)
M8: Generic GUI/Plot Module (described in MG_GUIplot)
M9: Trajectory Simulation Module (described in MG_SIM)

Figure 1: Use hierarchy among modules

[The uses relation is confusing. The input module should be “used by” other modules, not using them. I would think the input module would be lower in the hierarchy. I don’t know why the trajectory simulation module uses the GUI module? Maybe this will be clear to me (and you) when you write the MIS. —SS]

References

[You shouldn't do this manually, especially since you use BibTeX in the next section. The new references can be added to your .bib file and everything will be generated automatically. —SS]

- [1] Dynamics of multiple pendula
  http://wmii.uwm.edu.pl/~doliwa/IS-2012/Szuminski-2012-Olsztyn.pdf
- [2] Pendulum
  https://en.wikipedia.org/wiki/Pendulum
- [3] Pendulum (mathematics)
  https://en.wikipedia.org/wiki/Pendulum_(mathematics)
- [4] Double Pendulum
  https://en.wikipedia.org/wiki/Double_pendulum
- [4] Differential-Algebraic Equations by Taylor Series
  http://www.cas.mcmaster.ca/~nedialk/daets/
- [5] Multi-body Lagrangian Simulations
  https://www.youtube.com/channel/UCCuLchOx0W0yoNE9KOCYlVQ
- [6] The double pendulum: Lagrangian formulation
  https://diego.assencio.com/?index=1500c66ae7ab27bb0106467c68feebc6
- [7] Poincar map
  https://en.wikipedia.org/wiki/Poincar%C3%A9_map
- [8] D. L. Parnas, "On the criteria to be used in decomposing systems into modules", Comm. ACM, vol. 15, pp. 1053-1058, December 1972.
- [9] D. Parnas, P. Clement, and D. M. Weiss, "The modular structure of complex systems", in International Conference on Software Engineering, pp. 408-419, 1984.
- [10] D. L. Parnas, "Designing software for ease of extension and contraction," in ICSE '78: Proceedings of the 3rd international conference on Software engineering, (Piscataway, NJ, USA), pp. 264-277, IEEE Press, 1978.
- [11] Smith and Lai(2005); Smith et al. (2007).
  See bib file that doesn't work for me for full citation.

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.