CAS 741: Module Interface Specification

Dynamical Systems: MPSim

Karol Serkis
serkiskj@mcmaster.ca
GitHub: karolserkis

December 19, 2018

1 Revision History

Date	Version	Notes
November 26, 2018	1.0	First full draft for submission
December 17, 2018	2.0	All GitHub issues and comments addressed
December 18, 2018	3.0	MIS & bibliography fixed, & program name fixed

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at CAS-741-Pendula SRS
The symbols are listed in alphabetical order.

symbol	description
A	Assumption
DD	Data Definition
GD	General Definition
GS	Goal Statement
IM	Instance Model
LC	Likely Change
NF	Non-Functional Requirement
MIS	Module Interface Specification
PS	Physical System Description
R	Requirement
SRS	Software Requirements Specification
T	Theoretical Model

Contents

1 Revision History **i**

2 Symbols, Abbreviations and Acronyms **ii**

3 Introduction **1**

4 Notation **1**

5 Module Decomposition **1**

6 MIS of MPSim Control Module (M2) **2**
- 6.1 Module . . . 2
- 6.2 Uses . . . 2
- 6.3 Syntax . . . 3
  - 6.3.1 Exported Constants . . . 3
  - 6.3.2 Exported Access Programs . . . 3
- 6.4 Semantics . . . 3
  - 6.4.1 State Variables . . . 3
  - 6.4.2 Environment Variables . . . 3
  - 6.4.3 Assumptions . . . 3
  - 6.4.4 Access Routine Semantics . . . 3
  - 6.4.5 Local Functions . . . 4

7 MIS of MPSim GUI (M3) **5**
- 7.1 Module . . . 5
- 7.2 Uses . . . 5
- 7.3 Syntax . . . 5
  - 7.3.1 Exported Constants . . . 5
  - 7.3.2 Exported Access Programs . . . 5
- 7.4 Semantics . . . 5
  - 7.4.1 State Variables . . . 5
  - 7.4.2 Environment Variables . . . 6
  - 7.4.3 Assumptions . . . 6
  - 7.4.4 Access Routine Semantics . . . 6
  - 7.4.5 Local Functions . . . 7

8 MIS of User Input Module (M4) **7**
- 8.1 Module . . . 7
- 8.2 Uses . . . 7
- 8.3 Syntax . . . 7
  - 8.3.1 Exported Constants . . . 7
  - 8.3.2 Exported Access Programs . . . 7

8.4 Semantics . . . 8
8.4.1 State Variables . . . 8
8.4.2 Environment Variables . . . 8
8.4.3 Assumptions . . . 8
8.4.4 Access Routine Semantics . . . 8
8.4.5 Local Functions . . . 9

9 MIS of Lagrangian Module (M5) 9
9.1 Module . . . 9
9.2 Uses . . . 9
9.3 Syntax . . . 10
9.3.1 Exported Constants . . . 10
9.3.2 Exported Access Programs . . . 10
9.4 Semantics . . . 10
9.4.1 State Variables . . . 10
9.4.2 Environment Variables . . . 10
9.4.3 Assumptions . . . 10
9.4.4 Access Routine Semantics . . . 10
9.4.5 Local Functions . . . 11

10 MIS of Data Structure Module (M7) 11
10.1 Module . . . 11
10.2 Uses . . . 11
10.3 Syntax . . . 12
10.3.1 Exported Constants . . . 12
10.3.2 Exported Access Programs . . . 12
10.4 Semantics . . . 12
10.4.1 State Variables . . . 12
10.4.2 Environment Variables . . . 12
10.4.3 Assumptions . . . 12
10.4.4 Access Routine Semantics . . . 12
10.4.5 Local Functions . . . 13

11 MIS of Generic GUI/Plot Module (M8) 13
11.1 Module . . . 13
11.2 Uses . . . 14
11.3 Syntax . . . 14
11.3.1 Exported Constants . . . 14
11.3.2 Exported Access Programs . . . 14
11.4 Semantics . . . 14
11.4.1 State Variables . . . 14
11.4.2 Environment Variables . . . 14
11.4.3 Assumptions . . . 14

11.4.4 Access Routine Semantics 14
11.4.5 Local Functions 15

12 MIS of Trajectory Simulation Module (M9) 15
12.1 Module 15
12.2 Uses 15
12.3 Syntax 16
12.3.1 Exported Constants 16
12.3.2 Exported Access Programs 16
12.4 Semantics 16
12.4.1 State Variables 16
12.4.2 Environment Variables 16
12.4.3 Assumptions 16
12.4.4 Access Routine Semantics 16
12.4.5 Local Functions 17

3 Introduction

The following document details the Module Interface Specifications for the solution for a Multi-Pendulum Simulation (MPSim) program and tracking the chaotic motion of the system in a simulation.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at:
https://github.com/karolserkis/CAS-741-Pendula.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

(Karol) I am also using https://proofwiki.org/wiki/Symbols:R for notation reference

The following table summarizes the primitive data types used by MPSim and the symbols used in this document. The choice of symbols was made to be consistent with calculus, ordinary differentials (ODE), the Lagrangian, kinematics etc. The standard mathematical spaces are used (e.g. $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$, etc.)

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of MPSim uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, MPSim uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

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

Table 1: Module Hierarchy

6 MIS of MPSim Control Module (M2)

Execution flow of MPSim . Calls the different modules in the appropriate order. The general steps of the module are:

1. Read in and verify mps_input.
2. Calculate Lagrangian module, iterate over the whole time frame (use data structure).
3. Generate a Plot
4. Return the trajectory plot output as per the output/simulation module specifications.

6.1 Module

mps_control

6.2 Uses

mps_gui (M3, Section 7)
mps_input (M4, Section 8)

Processing modules: [what are processing modules? I'm not sure about this terminology —SS]
mps_lagrang (M5, Section 9)
mps_datastruct (M7, Section 11)
generic_plot (M8, Section 12), mps_trajplotsim (M9, Section 13)

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
run_mps	int, str	None	None

6.4 Semantics

The MPSim is designed to have the input and initialization controlled by the user directly through the GUI. The MPSim Control Module uses the events in the MPSim GUI to use the processing modules in order.

6.4.1 State Variables

- mps_input_init : init (see Section 8)
- mps_datastruct_init : init (see Section 11)
- mps_lagrang : calc (see Section 9)

[init is not a type, either is calc? How am I supposed to read this? State variables are specified as a name and a type. —SS]

6.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

6.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

6.4.4 Access Routine Semantics

run_mps(mps_input) [This doesn't match the int, str specified above? —SS]:

- transition: The Multi Pendulum Simulation is initialized and the data structure is passed through the Lagrangian module for calculation (The result is then passed to the plot/simulation/output modules.)

- output: None [if appropriate —SS]
- exception: None [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

6.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

[A new page between modules makes the document a more convenient reference document. —SS]

7 MIS of MPSim GUI (M3)

[Use labels for cross-referencing —SS]

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hyper-links to external documents. —SS]

Methods to interact with user and display & run MPSim . 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.

7.1 Module

mps_gui

7.2 Uses

mps_datastruct (M7, Section 11)
generic_plot (M8, Section 12), mps_trajplotsim (M9, Section 13)

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
mps_input	int, str	-	-
gui_trajsim	-	GUI object	-
gui_plot	-	GUI object (Poincaré)	-

7.4 Semantics

The MPSim process flow is initialized buy user through the GUI. The user triggered the events that start the selected processing step.

7.4.1 State Variables

None

7.4.2 Environment Variables

Plot_TrajSim: GUI object
button_Input: GUI object
button_TrajSim: GUI object

[More words to describe these environment variables would be helpful. —SS]

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

7.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

7.4.4 Access Routine Semantics

mps_input():

- transition: Trigger mps_input when button_Input pressed.
- output: [if appropriate —SS]
- exception: WarnOutOfBounds (warn when mps_input are invalid) [if appropriate —SS]

gui_trajsim():

- transition: Trigger gui_trajsim when button_TrajSim pressed.
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

gui_plot():

- transition: Trigger gui_plot when button_Poincaré pressed.
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

7.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

8 MIS of User Input Module (M4)

[Use labels for cross-referencing —SS]
[You can reference SRS labels, such as R??. —SS]
[It is also possible to use LaTeXfor hyperlinks to external documents. —SS]

The format and structure of the input data. Loads, verifies and stores the input data into the appropriate data and object structure.

8.1 Module

mps_input

8.2 Uses

mps_gui (M3, Section 7)
mps_datastruct (M7, Section 11) [The modules in the uses section should actually be used in the specification. I don't see where these are actually invoked. —SS]

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
verify_Input	int, str	-	-
data_struct	int, str	-	-
gui_trajsim	-	GUI object	-
gui_plot	-	GUI object	-

8.4 Semantics

8.4.1 State Variables

data : object

8.4.2 Environment Variables

Plot_TrajSim: GUI object
[Describe this enviro variable. Are you taking the data fields from corresponding fields in the GUI object? —SS] [You have really integrated the GUI into your design. It is better if you almost don't mention the GUI with the idea that the modules can be implemented by any interface. Maybe a model view controller pattern would work in your case? —SS]

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

8.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

8.4.4 Access Routine Semantics

verify_Input():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: Check if positive integer

data_struct(mps_input):

- transition: mps_input is the user input that is entered into the GUI. The input is used to initialize the data structure. [What is the data structure? The MIS is where you make decisions about this. —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

gui_trajsim(mps_datastruct, mps_input):

- transition: mps_datastruct is a module that takes the mps_input and thus the program gui_trajsim will be prepared for the Lagrangian and Hamiltonian calculations done by their corresponding modules.

- output: [if appropriate —SS]
- exception: [if appropriate —SS]

gui_plot(mps_datastruct, mps_input):

- transition: mps_datastruct is a module that takes the mps_input and thus the program gui_plot will be prepared for the Lagrangian calculations done by their corresponding modules.
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

8.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

9 MIS of Lagrangian Module (M5)

[Use labels for cross-referencing —SS]

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hyperlinks to external documents. —SS]

Lagrangian calculation/algorithm. Calculates the variation of Lagrangian for each pendulum in the chain (each element of the array).

9.1 Module

mps_lagrang

9.2 Uses

mps_input (M4, Section 8)

Processing modules:
mps_lagrang (M5, Section 9)

mps_datastruct (M7, Section 11)

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
mps_input	int	-	-
calc_lagrang	int	float(python)	-

9.4 Semantics

9.4.1 State Variables

9.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

9.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

9.4.4 Access Routine Semantics

mps_input():

- transition: mps_input passed to calc_lagrang (which in turn the module utilizes mps_datastruct).
- output: [if appropriate —SS]
- exception: WarnOutOfBounds (warn when mps_input are invalid) [if appropriate —SS]

calc_lagrang(mps_input):

- transition: Utilizing the mps_datastruct the mps_input is calculated and passed to the gui_trajsim and gui_plot [if appropriate —SS]
- output: To gui_trajsim and gui_plot [if appropriate —SS]

- exception: WarnOutOfBounds (warn when mps_input are invalid) [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

9.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

[For the Lagrangian and Hamiltonian modules, I expected to see equations, like those from your SRS. —SS]

10 MIS of Data Structure Module (M7)

[Use labels for cross-referencing —SS]

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hyperlinks to external documents. —SS]

Data format for an image. Provides convenient format to store, read and manipulate all OpenGL elements. Also the points to plot on a graph (where the image is constructed by an python libraries).

10.1 Module

mps_datastruct

10.2 Uses

mps_gui (M3, Section 7)
mps_input (M4, Section 8)

Processing modules:
mps_lagrang (M5, Section 9)
mps_datastruct (M7, Section 11)

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
calc_lagrang	int, str	long, str	-
calc_hamil	long, str	long, str	-
mps_gui	GUI object	-	-

10.4 Semantics

10.4.1 State Variables

[the data structure doesn't have state variables? —SS]

10.4.2 Environment Variables

Plot_TrajSim: GUI object
Plot_Poincaré: GUI object

[Why does the data structure use gui objects? As mentioned previously, the GUI connects to too many modules. Most modules should be independent of the interface. —SS]

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

10.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

10.4.4 Access Routine Semantics

calc_lagrang(mps_input):

- transition: Utilizing the mps_datastruct the mps_input is calculated and passed to the gui_trajsim and gui_plot [if appropriate —SS]
- output: To gui_trajsim and gui_plot [if appropriate —SS]
- exception: WarnOutOfBounds (warn when mps_input are invalid) [if appropriate —SS]

calc_hamil(mps_input):

- transition: Utilizing the mps_datastruct the mps_input is calculated and passed to the gui_trajsim and gui_plot [if appropriate —SS]
- output: To gui_trajsim and gui_plot [if appropriate —SS]
- exception: WarnOutOfBounds (warn when mps_input are invalid) [if appropriate —SS]

mps_gui(mps_datastruct, gui_trajsim, gui_plot):

- transition: Utilizing the mps_datastruct the mps_input is calculated and passed to the gui_trajsim and gui_plot and thus the program gui_trajsim will be prepared for the Lagrangian and Hamiltonian calculations done by their corresponding modules.
- output: To gui_trajsim and gui_plot [if appropriate —SS]
- exception: WarnOutOfBounds (warn when mps_input are invalid) [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

10.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

11 MIS of Generic GUI/Plot Module (M8)

[Use labels for cross-referencing —SS]

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hyperlinks to external documents. —SS]

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

11.1 Module

generic_plot

11.2 Uses

Hardware-Hiding (M1)

11.3 Syntax

11.3.1 Exported Constants

None

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
gui_trajsim	GUI object	GUI object	-
gui_plot	GUI object	GUI object	-

11.4 Semantics

11.4.1 State Variables

11.4.2 Environment Variables

Plot_TrajSim: GUI object
[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

11.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

11.4.4 Access Routine Semantics

gui_trajsim():

- transition: [if appropriate —SS]
- output: To mps_gui(mps_datastruct, gui_trajsim, gui_plot) or the generic_plot module [if appropriate —SS]
- exception: WarnOutOfBounds (warn when gui_trajsim are invalid) [if appropriate —SS]

gui_plot(mps_datastruct, mps_input):

- transition: mps_datastruct is a module that takes the mps_input and thus the program gui_plot will be prepared for the Lagrangian and Hamiltonian calculations done by their corresponding modules.
- output: [if appropriate —SS] [if it isn't appropriate, you should remove this field. —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

11.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

12 MIS of Trajectory Simulation Module (M9)

[Use labels for cross-referencing —SS]

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hyperlinks to external documents. —SS]

Algorithm to simulate the plot trajectory of the chain of pendula. Simulates the chain of pendula in space using the Lagrangian and the Hamiltonian.

12.1 Module

mps_trajplotsim

12.2 Uses

mps_gui (M3, Section 7)
mps_input (M4, Section 8)

Processing modules:
mps_lagrang (M5, Section 9), mps_hamil (M6, Section 10)

12.3 Syntax

12.3.1 Exported Constants

None

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
MPS_trajsim	GUI object	GUI object	WarnOutOfBounds

12.4 Semantics

12.4.1 State Variables

data : object

12.4.2 Environment Variables

Plot_TrajSim: GUI object

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

12.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

12.4.4 Access Routine Semantics

MPS_trajsim():

- transition:
    1. Read in and verify mps_input.
    2. Calculate Lagrangian module calc_lagrang, iterate over the whole time frame (use data structure).
    3. Calculate Hamiltonian module calc_hamil, iterate over the whole time frame (use data structure).
    4. Return the trajectory plot output as per the output/simulation module specifications.
- output: [if appropriate —SS]

- exception: WarnOutOfBounds (warn when mps_input or calculations of the data structure are invalid) [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

12.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.