SE 2AA4, CS 2ME3 (Introduction to Software Development)

Winter 2018

# 18 Modules with External Interaction (H&S Section 7.4)

Dr. Spencer Smith

Faculty of Engineering, McMaster University

February 15, 2018

McMaster University

# 18 Modules with External Interaction (H&S Section 7.4)

- Administrative details
- Modules with external interaction
- MIS for GUI module
- MIS for vector space module
- MIS for SWHS

# Administrative Details

- Assignment 2
  - Part 1: February 20, 2018
  - Partner Files: February 26, 2018
  - Part 2: March 2, 2018
- Midterm exam
  - Wednesday, February 28, 7:00 pm
  - 90 minute duration
  - This week's tutorial will cover a sample midterm

# Motor

With our current MIS framework, how would you represent calling the access program `toggleMotor` to either turn a motor on or off, depending on its current state?

A. Introduce a state variable `powerOn` of type Boolean

B. Introduce an ADT for a motor type and pass an instance of this type as an argument

C. Return a Boolean that is True if the motor is on and False otherwise

D. Use an environment variable, like: motor: $\mathbb{B}$

# Modules with External Interaction

- In general, some modules may interact with the environment or other modules
- Environment might include the keyboard, the screen, the file system, motors, sensors, etc.
- Sometimes the interaction is informally specified using prose (natural language)
- Can introduce an environment variable
  - Name, type
  - Interpretation
- Environment variables include the screen, the state of a motor (on, direction of rotation, power level, etc.), the position of a robot

# External Interaction Continued

- Some external interactions are hidden
  - Present in the implementation, but not in the MIS
  - Example
    - OS memory allocation calls
    - `import math` for assignments
- External interaction described in the MIS
  - Naming access programs of the other modules
  - Specifying how the other module's state variables are changed
  - The MIS should identify what external modules are used

# MIS for GUI Modules

- Could introduce an environment variable
- window: sequence [RES_H][RES_V] of pixelT
  - Where window[r][c] is the pixel located at row r and column c, with numbering zero-relative and beginning at the upper left corner
  - Would still need to define pixelT
- Could formally specify the environment variable transitions
- More often it is reasonable to specify the transition in prose
- In some cases the proposed GUI might be shown by rough sketches

# Display Point Masses Module Syntax

## Exported Access Programs

| Routine name | In | Out | Exc. |
|---|---|---|---|
| DispPointMassesApplet | | DispPointMassesApplet | |
| paint | | | |

Steps

1. DispPointMassessApplet first builds a list of point masses in the ListPointMasses abstract object (the list is explicitly given in the spec)
2. paint then displays the point masses on the screen, as circles with the centre at the coordinates for the point mass and the radius equal to the mass

What external interactions do we have? What environment variables do we need? What if we read the data in from a file?

# Display Point Masses Module Semantics

**Environment Variables**

*win* : 2D sequence of pixels displayed within a web-browser

DispPointMassesApplet():

- transition: The state of the abstract object ListPointMasses is modified as follows:
  ListPointMasses.init()
  ListPointMasses.add(0, PointMassT(20, 20, 10)
  ListPointMasses.add(1, PointMassT(120, 200, 20)
  ...

paint():

- transition *win* := Modify window so that the point masses in ListPointMasses are plotted as circles. The centre of each circles should be the corresponding x and y coordinates and the radius should be the mass of the point mass.

# Vector Space Module

## Exported Access Programs

| **Routine name** | **In** | **Out** | **Exceptions** |
|---|---|---|---|
| new vectorT | real $\rightarrow$ real | vectorT | |
| getf | | real $\rightarrow$ real | |
| eval | real, real, integer | seq of real | deltaNeg, nstepsNeg |
| evalPrint | real, real, integer | | deltaNeg, nstepsNeg |

# Vector Module Semantics

## Environment Variables

*screen* : two dimensional sequence of positions on the screen, where each position holds a character

## State Variables

$f$: real $\rightarrow$ real

## Access Routine Semantics

eval $(startx, deltax, nsteps)$:

- output: $out := < f(startx), f(startx + deltax), f(startx + 2 \cdot deltax), ..., f(startx + nsteps \cdot deltax) >$
- exception: $exc := ((deltax < 0) \Rightarrow \text{deltaNeg} | (nsteps < 0) \Rightarrow \text{nstepsNeg})$

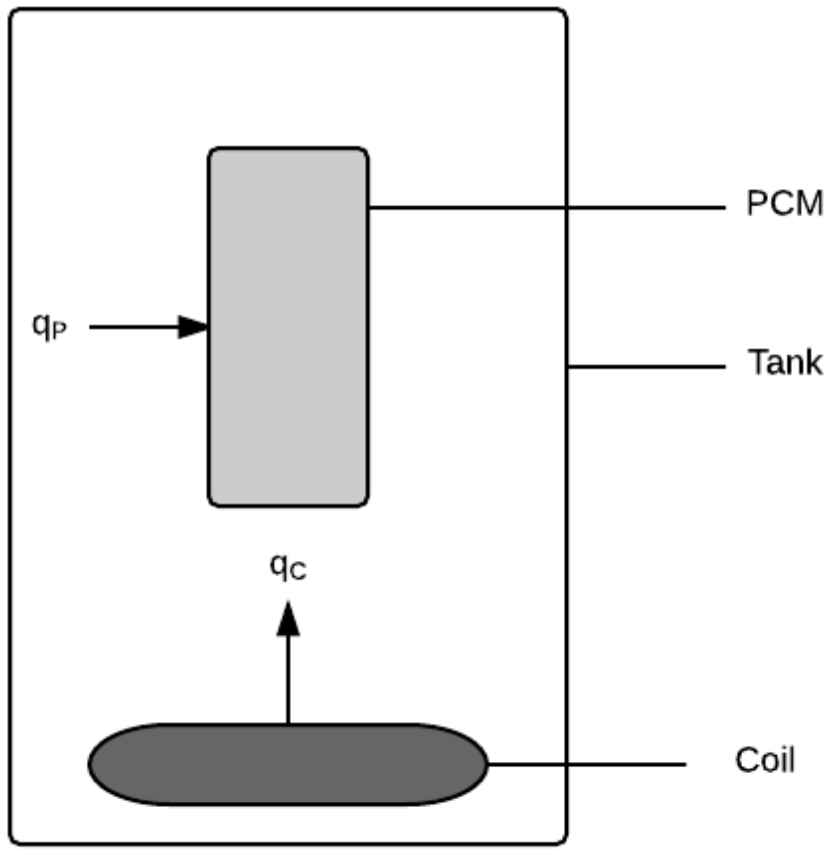
# Vector Module Semantics Continued

evalPrint (*startx*, *deltax*, *nsteps*):

- transition: The state of *screen* is modified so that the sequence returned by eval (*startx*, *deltax*, *nsteps*) is displayed. All values in the 2D sequence screen will be white space, except for the coordinates screen[x][f(x)], where x corresponds to the horizontal distance across the screen. # *We could also introduce the notion of scaling.*
- exception: $exc := ((deltax < 0) \Rightarrow \text{deltaNeg} | (nsteps < 0) \Rightarrow \text{nstepsNeg})$

# Solar Water Heating System Example

- https://github.com/smiths/swhs
- Solve ODEs for temperature of water and PCM
- Solve for energy in water and PCM
- Generate plots



# Anticipated Changes?

## What are some anticipated changes?

Hint: the software follows the Input $\rightarrow$ Calculate $\rightarrow$ Output design pattern

# Anticipated Changes

- The specific hardware on which the software is to run
- The format of the initial input data
- The format of the input parameters
- The constraints on the input parameters
- The format of the output data
- The constraints on the output results
- How the governing ODEs are defined using the input parameters
- How the energy equations are defined using the input parameters
- How the overall control of the calculations is orchestrated
- The implementation of the sequence data structure
- The algorithm used for the ODE solver
- The implementation of plotting data

# Module Hierarchy by Secrets

| **Level 1** | **Level 2** |
|---|---|
| Hardware-Hiding Module | |
| Behaviour-Hiding Module | Input Parameters Module<br>Output Format Module<br>Output Verification Module<br>Temperature ODEs Module<br>Energy Equations Module<br>Control Module<br>Specification Parameters Module |
| Software Decision Module | Sequence Data Structure Module<br>ODE Solver Module<br>Plotting Module |

# Example Modules from SWHS

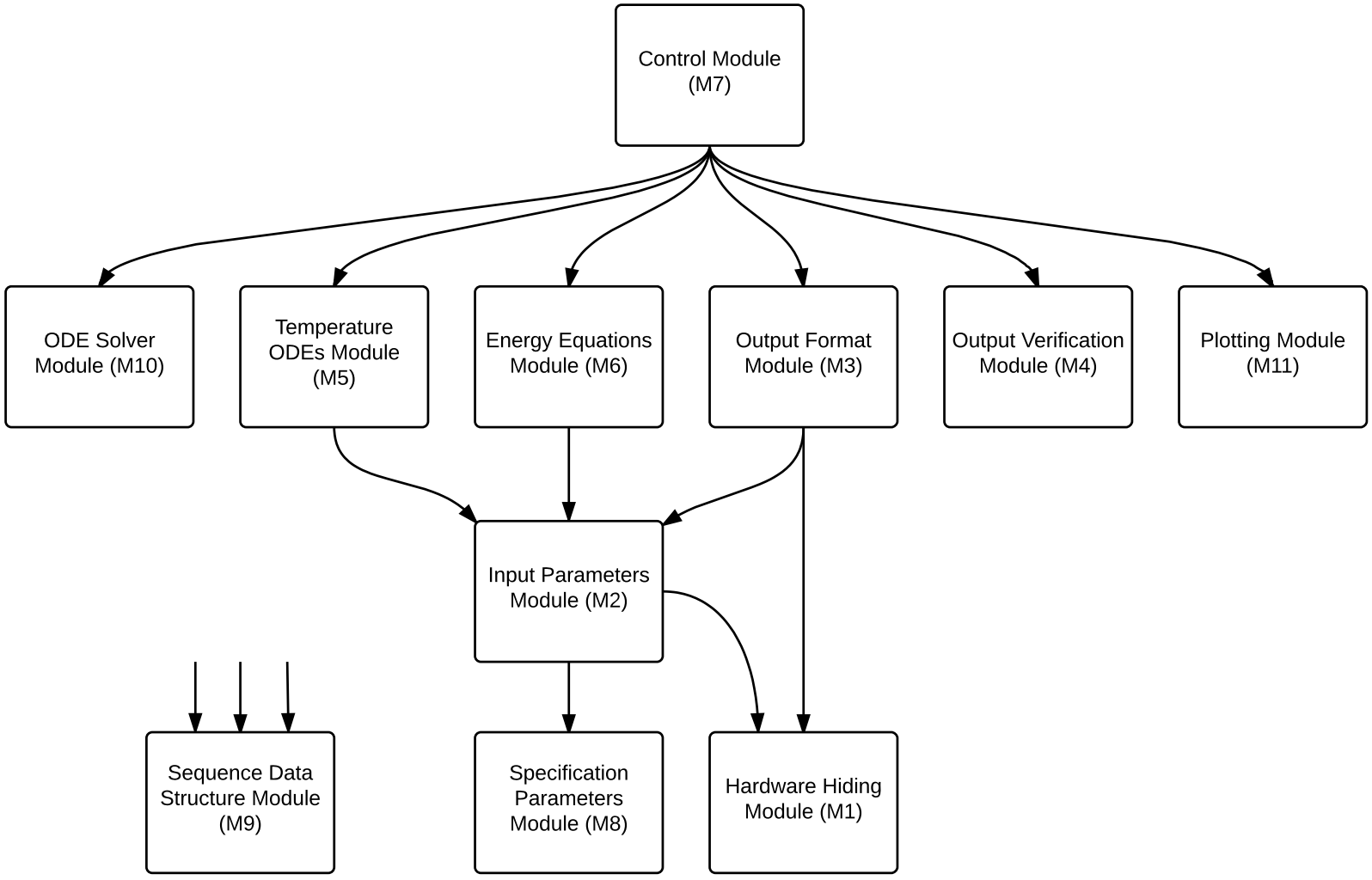
## ODE Solver Module

Secrets: The algorithm to solve a system of first order ODEs.

Services: Provides solvers that take the governing equation, initial conditions, and numerical parameters, and solve them.

Implemented By: Matlab

# SWHS Uses Hierarchy




MIS document