CAS 741, CES 741 (Development of Scientific Computing Software)

Fall 2019

14 Module Interface Specification (MIS)

Dr. Spencer Smith

Faculty of Engineering, McMaster University

October 28, 2019

Module Interface Specification (MIS)

- Administrative details
- Questions?
- Finish previous day slides
- Module guide example
- Integration testing
- Mathematical review ([4] and separate slides)
  - Multiple assignment statement
  - Conditional rules
  - etc.
- MIS overview
- Modules with external interaction
- Abstract objects
- Abstract data types
- Generic MIS
- Inheritance

Administrative Details: Report Deadlines

System VnV Plan	Week 08	Oct 28
MG + MIS	Week 10	Nov 25
Final Documentation	Week 14	Dec 9

- The written deliverables will be graded based on the repo contents as of 11:59 pm of the due date
- If you need an extension, please ask
- Two days after each major deliverable, your GitHub issues will be due
- Domain expert code due 1 week after MIS deadline

Administrative Details: Presentations

MG + MIS Syntax Present	Week 9	Week of Nov 4
MIS Semantics Present	Week 11	Week of Nov 18
Unit VnV or Impl. Present	Week 12/13	Week of Nov 28

- Informal presentations with the goal of improving everyone's written deliverables
- Domain experts and secondary reviewers (and others) will ask questions (listed in Repos.xlsx file)

Administrative Details: Presentation Schedule

- MG + MIS Syntax Present
  - Monday: **Deema, Bo, ?**
  - Thursday: **Sasha, ?, ?**
- MIS Syntax + Semantics Present
  - Monday: Zhi, Peter, *Sasha*
  - Thursday: Sharon, Ao, ?
- Unit VnV Plan or Impl. Present
  - Monday: Bo, Sasha, ?
  - Thursday: Zhi, Peter, Ao, ?

Optional presentation in italics.
Room for more volunteers. :-)

Questions?

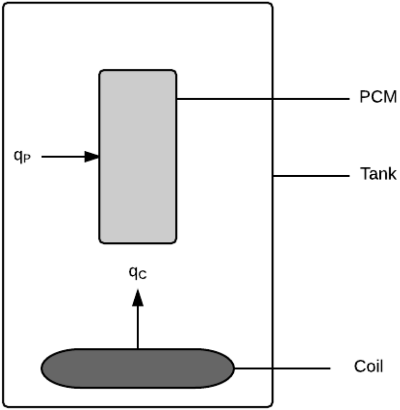
- Questions about Module Guide and the presentation?

Finish Previous Day's Discussion

- Static Definition of Uses Relation
- Module Guide
- MG Template
- MG Verification
- OO versus modular

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 Output Format Module Temperature ODEs Module Energy Equations Module Control Module Specification Parameters
Software Decision Module	Sequence Data Structure Module ODE Solver Module Plotting Module

Table: Module Hierarchy

Example Modules from SWHS

Hardware Hiding Modules

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

Example Modules from SWHS

Input Parameters Module

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).

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.

Implemented By: SWHS

Example Modules from SWHS

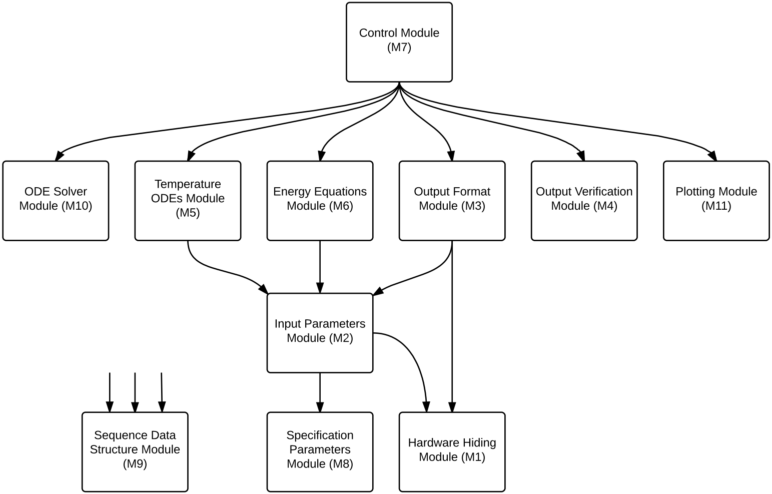
ODE Solver Module

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

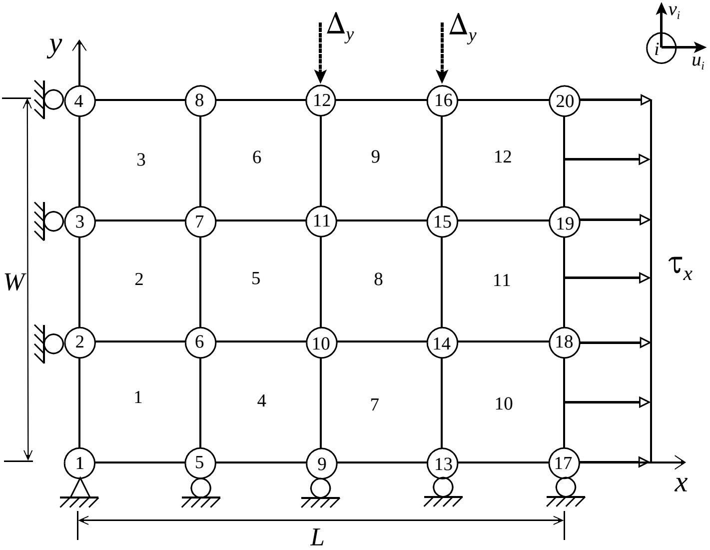
Services: Solves an ODE using the governing equation, initial conditions, event function and numerical parameters.

Implemented By: Matlab

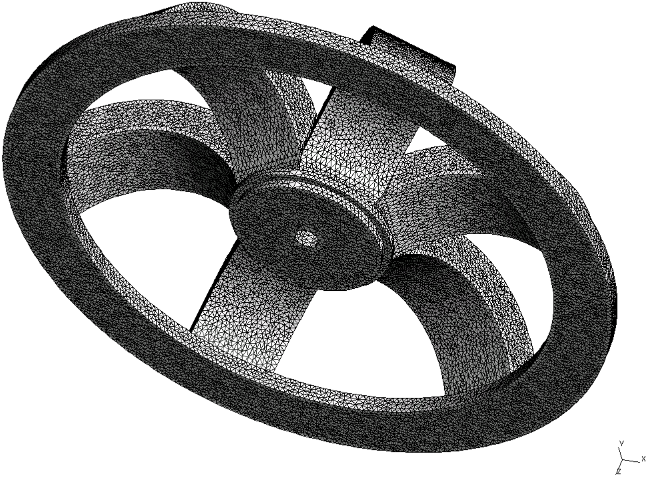
SWHS Uses Hierarchy (approximately)



Mesh Generator Example



Mesh Generator Complex Circular Geometry



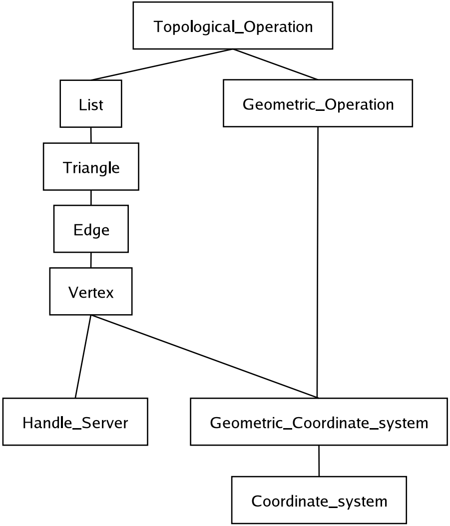
Mesh Generator Example: Design Goals

- Independent and flexible representation for each mesh entity
- Complete separation of geometric data from the topology
- The mesh generator should work with different coordinate systems
- A flexible data structure to store sets of vertices, edges and triangles
- Different mesh generation algorithms with a minimal amount of local changes

Example Mesh Gen Modular Decomposition

Link

Another Mesh Generator Uses Hierarchy [2]



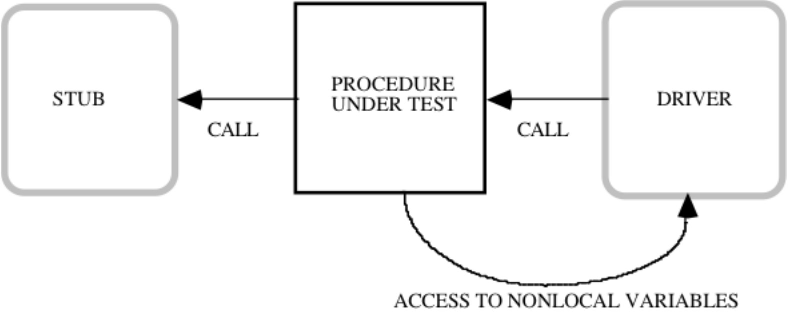
Module Testing

Is it possible to begin testing before all of the modules have been implemented when there is a use relation between modules?

Module Testing [3]

- Scaffolding needed to create the environment in which the module should be tested
- Stubs - a module used by the module under test
- Driver - module activating the module under test

Testing a Functional Module [3]



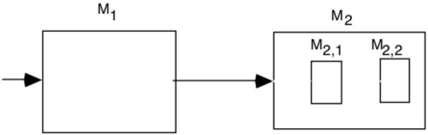
Integration Testing

- Big-bang approach
  - First test individual modules in isolation
  - Then test integrated system
- Incremental approach
  - Modules are progressively integrated and tested
  - Can proceed both top-down and bottom-up according to the USES relation

Integration Testing and USES relation

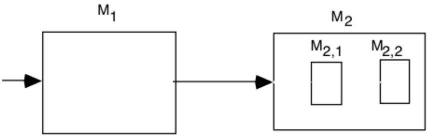
- If integration and test proceed bottom-up only need drivers
- Otherwise if we proceed top-down only stubs are needed

Example [3]



- M_1 USES M_2 and M_2 IS_COMPOSED_OF $\{M_{2,1}, M_{2,2}\}$
- In what order would you test these modules?

Example [3]



- M_1 USES M_2 and M_2 IS_COMPOSED_OF $\{M_{2,1}, M_{2,2}\}$
- Case 1
  - Test M_1 providing a stub for M_2 and a driver for M_1
  - Then provide an implementation for $M_{2,1}$ and a stub for $M_{2,2}$
- Case 2
  - Implement $M_{2,2}$ and test it by using a driver
  - Implement $M_{2,1}$ and test the combination of $M_{2,1}$ and $M_{2,2}$ (i.e. M_2) by using a driver
  - Finally implement M_1 and test it with M_2 using a driver for M_1

Overview of MIS

- See Hoffman and Strooper [4]
- The MIS precisely specifies the modules observable behaviour - what the module does
- The MIS does not specify the internal design
- The idea of an MIS is inspired by the principles of software engineering
- Advantages
  - Improves many software qualities
  - Programmers can work in parallel
  - Assumptions about how the code will be used are recorded
  - Test cases can be decided on early, and they benefit from a clear specification of the behaviour
  - A well designed and documented MIS is easier to read and understand than complex code
  - Can use the interface without understanding details

Overview of MIS

- Options for specifying an MIS
  - Trace specification
  - Pre and post conditions specification
  - Input/output specification
  - Before/after specification - module state machine
  - Algebraic specification
- Best to follow a template

MIS Template

- Uses
  - Imported constants, data types and access programs
- Syntax
  - Exported constants and types
  - Exported functions (access routine interface syntax)
- Semantics
  - State variables
  - State invariants
  - Assumptions
  - Access routine semantics
  - Local functions
  - Local types
  - Local constants
  - Considerations

MIS Uses Section

- Specify imported constants
- Specify imported types
- Specify imported access programs
- The specification of one module will often depend on using the interface specified by another module
- When there are many modules the uses information is very useful for navigation of the documentation
- Documents the use relation between modules

MIS Syntax Section

- Specify exported constants
- Specify exported types
- Specify access routine names, the input and output parameter types and exceptions
- Show access routines in tabular form
  - Important design decisions are made at this point
  - The goal is to have the syntax match many implementation languages

Syntax of a Sequence Module

Exported Constants

MAX_SIZE = 100

Syntax of a Sequence Module Continued

Exported Access Programs

Routine name	In	Out	Exceptions
seq_init			
seq_add	integer, integer		FULL, POS
seq_del	integer		POS
seq_setval	integer, integer		POS
seq_getval	integer	integer	POS
seq_size		integer	

MIS Semantics Section

- State variables
  - Give state variable(s) name and type
  - State variables define the state space
  - If a module has state then it will have “memory”
- State invariant
  - A predicate on the state space that restricts the “legal” states of the module
  - After every access routine call, the state should satisfy the invariant
  - Cannot have a state invariant without state variables
  - Just stating the invariant does not “enforce” it, the access routine semantics need to maintain it
  - Useful for understandabilty, testing and for proof

Semantics Section Continued

- Local functions, local types and local constants
  - Declared for specification purposes only
  - Not available at run time
  - Helpful to make complex specifications easier to read
- Considerations
  - For information that does not fit elsewhere
  - Useful to tell the user if the module violates a quality criteria

Sequence MIS Semantics

State Variables

s: sequence of integer

State Invariant

$|s| \leq \mathrm{MAX\_SIZE}$

Assumptions

seq_init() is called before any other access program

Sequence MIS Semantics Continued

Access Routine Semantics

seq_init():

- transition: $s :=<>$
- exception: none

seq_add(i, p):

- transition: $s := s[0..i-1] || < p > || s[i..|s|-1]$
- exception:
  $exc := (|s| = \mathrm{MAX\_SIZE} \Rightarrow \mathrm{FULL} \mid i \notin [0..|s|] \Rightarrow \mathrm{POS})$

Access Routine Semantics Continued

seq_del(i):

- transition: $s := s[0..i-1] || s[i+1..|s|-1]$
- exception: $exc := (i \notin [0..|s|-1] \Rightarrow \mathrm{POS})$

seq_setval(i, p):

- transition: $s[i] := p$
- exception: $exc := (i \notin [0..|s|-1] \Rightarrow \mathrm{POS})$

seq_getval(i):

- output: $out := s[i]$
- exception: $exc := (i \notin [0..|s|-1] \Rightarrow \mathrm{POS})$

Access Routine Semantics Continued

seq_size():

- output: $out := |s|$
- exception: none

Exception Signaling

- Useful to think about exceptions in the design process
- Will need to decide how exception signalling will be done
  - A special return value, a special status parameter, a global variable
  - Invoking an exception procedure
  - Using built-in language constructs
- Caused by errors made by programmers, not by users
- Write code so that it avoid exceptions
- Exceptions will be particularly useful during testing

Assumptions versus Exceptions

- The assumptions section lists assumptions the module developer is permitted to make about the programmer's behaviour
- Assumptions are expressed in prose
- Use assumptions to simplify the MIS and to reduce the complexity of the final implementation
- Interface design should provide the programmer with a means to check so that they can avoid exceptions
- When an exceptions occurs no state transitions should take place, any output is *don't care*

References I

Jacques Carette, Mustafa ElSheikh, and W. Spencer Smith.
A generative geometric kernel.
In *ACM SIGPLAN 2011 Workshop on Partial Evaluation and Program Manipulation (PEPM'11)*, pages 53–62, January 2011.

Ahmed H. ElSheikh, W. Spencer Smith, and Samir E. Chidiac.
Semi-formal design of reliable mesh generation systems.
Advances in Engineering Software, 35(12):827–841, 2004.

References II

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.

References III

W. Spencer Smith and Wen Yu.
A document driven methodology for improving the quality of a parallel mesh generation toolbox.
Advances in Engineering Software, 40(11):1155–1167, November 2009.