Module Interface Specification for Measuring Microstructure Changes During Thermal Treatment

Team #30, ReSprint
Edwin Do
Joseph Braun
Timothy Chen
Abdul Nour Seddiki
Tyler Magarelli

January 18, 2023

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Jan 18 2023	1.1	Added MIS info for UserInputValidation, HardwareInput-Validation, and Calculation Modules
Jan 18 2023	1.2	Added state invariants

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at [give url —SS]

[Also add any additional symbols, abbreviations or acronyms —SS]

Contents

3 Introduction

The following document details the Module Interface Specifications for [Fill in your project name and description —SS]

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at .... [provide the url for your repo —SS]

4 Notation

[You should describe your notation. You can use what is below as a starting point. —SS]

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 $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Measuring Microstructure Changes During Thermal Treatment.

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 Measuring Microstructure Changes During Thermal Treatment 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, Measuring Microstructure Changes During Thermal Treatment 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	Input Parameters Output Format Output Verification Temperature ODEs Energy Equations Control Module Specification Parameters Module
Software Decision	Sequence Data Structure ODE Solver Plotting

Table 1: Module Hierarchy

6 MIS of [Module Name —SS]

[Use labels for cross-referencing —SS]

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

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

6.1 Module

[Short name for the module —SS]

6.2 Uses

6.3 Syntax

6.3.1 Exported Constants

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

6.4 Semantics

6.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —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

[accessProg —SS]():

- transition: [if appropriate —SS]
- 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]

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]

7 MIS of Calculation Module

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

7.1 Module

Calculation

7.2 Uses

7.2.1 Imported Types

HardwareInput:
($Voltage : real$; $Time : real$; $Temperature : real$; $Current : real$)

UserInput:
($SamplingRate : real$; $SampleLength : real$; $SampleWidth : real$; $Filename : string$; $Name : string$; $SampleName : string$; $Date : string$)

7.2.2 Imported Access Programs

getHardwareInput(): HardwareInput
getUserInput(): UserInput

7.3 Syntax

7.3.1 Exported Constants

N/A

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
getResistance()	-	Real	INVALID
getResistivity()	-	Real	INVALID

7.4 Semantics

7.4.1 State Variables

Resistance : The calculated resistance value (real)
Resistivity : The calculated resistivity value (real)
SampleArea : The calculated area of sample based on the length and width from user's input

7.4.2 State Invariants

Resistance ≥ 0
Resistivity ≥ 0
SampleArea ≥ 0

7.4.3 Environment Variables

N/A

7.4.4 Assumptions

We assume that the user may enter invalid values for inputs such as characters, empty spaces etc.. This will cause the program to throw and INVALID exception. This type of programmer error is also captured in the UserInputValidation Module to improve redundancy.

7.4.5 Access Routine Semantics

getResistance():

- transition: N/A
- output: out:= Resistance
- exception: $exc := Resistance \notin \mathbb{R} \vee Resistance < 0 \Rightarrow INVALID$

getResistivity():

- transition: N/A
- output: out := Resistivity
- exception: $exc := Resistivity \notin \mathbb{R} \vee Resistivity < 0 \Rightarrow INVALID$

7.4.6 Local Functions

findSampleArea(SampleLength, SampleWidth)

- transition: SampleArea := SampleLength x SampleWidth

calcResistance(voltage, current):

- transition: Resistance := voltage/current
- exception: $exc :=$
  $voltage \notin \mathbb{R} \vee voltage < 0$
  $\vee\, current \notin \mathbb{R} \vee current < 0$
  $\Rightarrow INVALID$

calcResistivity(Resistance, SampleArea, SampleLength):

- transition: Resistivity := (resistance x SampleArea)/SampleLength
- exception: $exc :=$
  $voltage \notin \mathbb{R} \vee voltage < 0$
  $\vee\, current \notin \mathbb{R} \vee current < 0$
  $\Rightarrow INVALID$

8 MIS of UserInputValidation Module

[Use labels for cross-referencing —SS]

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

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

8.1 Module

UserInputValidation

8.2 Uses

8.2.1 Imported Types

UserInput:
($SamplingRate$: $real$; $SampleLength$: $real$; $SampleWidth$: $real$; $Filename$: $string$; $Name$: $string$; $SampleName$: $string$; $Date$: $string$)

8.3 Syntax

8.3.1 Exported Constants

N/A

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
getUserInput()	-	ADT (UserInput)	INVALID

8.4 Semantics

8.4.1 State Variables

N/A

8.4.2 State Invariants

N/A

8.4.3 Environment Variables

N/A

8.4.4 Assumptions

We assume that the user may enter invalid values for inputs such as characters, empty spaces etc.. This will cause the program to throw and INVALID exception.

8.4.5 Access Routine Semantics

getUserInput():

- output: out:= UserInput
- exception: $exc := validateFileData \neq TRUE \lor$
  $validateSampleData \neq TRUE$
  $\Rightarrow INVALID$

8.4.6 Local Functions

valdiateFileData(Filename, Date, Name):

- output: out:= TRUE
- exception: $exc := Filename.type \neq STRING \lor$
  $Date.type \neq STRING \lor$
  $Name.type \neq STRING$
  $\Rightarrow INVALID$

validateSampleData(SamplingRate, SampleLength, SampleWidth):

- output: out:= TRUE
- exception: $exc := SamplingRate \notin \mathbb{R} \lor SamplingRate < 0 \lor$
  $SampleLength \notin \mathbb{R} \lor SampleLength < 0 \lor$
  $SampleWidth \notin \mathbb{R} \lor SampleWidth < 0$
  $\Rightarrow INVALID$

9 MIS of HardwareInputValidation Module

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

9.1 Module

HardwareInputValidation

9.2 Uses

9.2.1 Imported Types

HardwareInput:
($Voltage : real$; $Time : real$; $Temperature : real$; $Current : real$)

9.3 Syntax

9.3.1 Exported Constants

N/A

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
getHardwareInput()	-	ADT (HardwareInput)	INVALID

9.4 Semantics

9.4.1 State Variables

N/A

9.4.2 State Invariants

N/A

9.4.3 Environment Variables

N/A

9.4.4 Assumptions

N/A

9.4.5 Access Routine Semantics

getHardwareInput():

- output: out:= HardwareInput
- exception: $exc := validateParameters \neq TRUE \Rightarrow INVALID$

9.4.6 Local Functions

validateParameters(Voltage, Time, Current):

- output: out:= TRUE
- exception: $exc := Voltage < 0 \vee$
  $Time < 0 \vee Current < 0$
  $\Rightarrow INVALID$

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.

10 Appendix

[Extra information if required —SS]