Module Interface Specification for MISEG

Ao Dong

December 13, 2019

1 Revision History

Date	Version	Notes
Nov 25	1.0	Initial Draft
Dec 11	1.1	Modification according to feedback
Dec 12	1.2	Modification according to implementation

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/Ao99/MIA/blob/master/docs/SRS/CA.pdf

Contents

3 Introduction

The following document details the Module Interface Specifications for MISEG which is for medical image segmentation.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/Ao99/MIA.

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 $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$. This document has one modification to the original notations: the concatenation notation || can be used to build a new sequence from an existing sequence. Forexample, s2 := $||(x : \mathbb{N}|x \in s1 \cdot x + 1)$, where s1 = $\langle 1, 2, ..., 10\rangle$, then s2 = $\langle 2, 3, ..., 11\rangle$. We adopt the notation in this way assuming that the elements in a sequence are in order rather than random. Using it in this way is for convenience and practical reasons, but not for properly expanding the original notation in a mathematical way. [explain that the notation is for convenience. It is not proper mathematics; the order would of the elements is actually random. The notation is adopted for practical reasons. —SS] [Explanation added. —Author]

The following table summarizes the primitive data types used by MISEG.

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 (-∞, ∞)
boolean	boolean	a value in $\{true, false\}$

The following table summarizes other data types used by MISEG.

Data Type	Notation	Description
DICOM file	inputFile	a DICOM image file
DICOM frame	dcmFrame	a frame of image in a DICOM image file
image data	imageData	a data structure containing width, height and a sequence of pixel values
bitmap file	outputFile	an 8-bit 2D grayscale bitmap image file

The specification of MISEG 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, MISEG 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 Module	
Behaviour-Hiding Module	Input Module Output Module Optimal Thresholds Calculation Image Verification Constant Values Control Module
Software Decision Module	Sequence Data Structure Image Data Structure

Table 1: Module Hierarchy

6 MIS of Control Module

6.1 Module

main

6.2 Uses

Input in Section 8, ThresCal in Section 10, Output in Section 11

6.3 Syntax

6.3.1 Exported Constants

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
main	-	-	

6.4 Semantics

6.4.1 State Variables

None

6.4.2 Environment Variables

None

6.4.3 Assumptions

None

6.4.4 Access Routine Semantics

main():

- transition: use other modules by following these steps
  1. Get ($filenameIn$: string) and ($filenameOut$: string) from user
  2. Input.loadInput($filenameIn$) [You filename is sometimes in italics and sometimes not. You should be consistent. —SS] [I put it this way because it is a parameter. I changed all parameters to italics everywhere —Author]
  3. For ($j : \mathbb{N}$) from 0 to Input.numFrames, repeat the following steps,
  4. ThresCal.calculation(j)

5. Output.displayThresholds() [You do not actually have to compare with true. If ThresCal.isThresValid is a Boolean, it is already True or False. This comment applies elsewhere in this spec as well. —SS][I made some change so this part doesn't compare True or False. But for the rest of the document, I modifiedas you suggested. —Author]
6. Output.writeOutput($filenameOut$) [Our notation does not have an "else". You can just say True in place of "else". This same comment applies throughout this document. —SS][I made some change so this part doesn't compare True or False. But for the rest of the document, I modified as you suggested. —Author]

- output: none
- exception: none

[Your exception covers the same ground as given in your pseudo code. You shouldn't really repeat this here, especially since it is ambiguous in the wayyou have worded it. I suggest you replace the exception information you have with a note that points the reader to the transition field for the definition ofthe exception behaviour. —SS] [I've moved all exception to other modules, so not anymore a problem here, but I'll also make sure not a problem in other parts. —Author]

6.4.5 Local Functions

None

7 MIS of Constant Values

7.1 Module

Constants

7.2 Uses

None

7.3 Syntax

7.3.1 Exported Constants

numsThres: set of $\mathbb{N}$
numsThres := {1, 2, 3}

[Are these constants all necessary? Are the bounds really more related to available memory? —SS] [I removed bounds for images. Now images are valid with width and height greater than 0 —Author] [For the choices, I suggest that you introduce an enumerated type (via an exported type, like the department names in the student allocation example). The names choice1 and choice2 do not really tell the reader anything. Can you give them more meaningful names? —SS] [I only kept a set of natural number here. When the user's choice is in this set, it's valid —Author]

7.3.2 Access Routine Semantics

N/A

8 MIS of Input Module

8.1 Module

Input

8.2 Uses

ImageData 9, ImageVerify in Section 12

8.3 Syntax

8.3.1 Exported Constants

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
loadInput	s: string	-	badInputFileName, badInputFile-Format
verifyInput		-	emptyFrameJ
loadedImages	-	sequence of imageData	
numFrames	-	$\mathbb{N}$	
isLoaded	-	sequence of boolean	

8.4 Semantics

8.4.1 State Variables

loadedImages: sequence of imageData
numFrames: $\mathbb{N}$
isLoaded: sequence of boolean

8.4.2 Environment Variables

inputFile: a .dcm or .dcm30 DICOM medical image file

8.4.3 Assumptions

The data type String has a method parseToNum() to parse a string (such as "1") to an $\mathbb{N}$ (such as 1).

8.4.4 Access Routine Semantics

Input.loadedImages:

- output: *out* := loadedImages
- exception: none

Input.numFrames:

- output: *out* := numFrames
- exception: none

Input.isLoaded:

- output: *out* := isLoaded
- exception: none

loadInput(s):

- transition: The filename s is first associated with the file f. inputFile is used to modify the state variables using the following procedural specification:
  1. Read the inputFile.
  2. numFrames := information from inputFile
  3. loadedImages := $||(f : \text{dcmFrame} \mid f \in \text{inputFile} \cdot \text{dcmToImage}(f))$
  4. verifyInput()
- output: none
- exception: $exc :=$
  a file name s cannot be found $\implies$ badInputFileName
  the format of inputFile is incorrect $\implies$ badInputFileFormat

verifyInput():

- transition: This function modifies the state variables using the following procedural specification:
  1. isLoaded := $||(img : \text{imageData} \mid img \in \text{loadedImages} \cdot \text{ImageVerify.verifyImageData}(img))$
  2. Initiate a local variable cnt: $\mathbb{N}$
  3. cnt := $+(b : \text{boolean} \mid b \in \text{isLoaded} \cdot (b \implies 1 \mid \text{True} \implies 0)$
  4. print "cnt image frames have been loaded".

- output: none
- exception: $exc :=$
  $\neg$ ImageVerify.verifyImageData(loadedImages[j]) $\implies$ emptyFrameJ

[You have a parallel data structure for loadedImages and isLoaded. They are two separate sequences. They both need to be indexed separately. What about having the index in this module. You can just give an index and it will return an image, as long as the image is loaded. There could be an exception otherwise. Something to think about. —SS] [I don't quite understand this comment here. It seems like what I've already being doing. For now —Author]

8.4.5 Local Functions

dcmToImage: dcmFrame $\rightarrow$ imageData
dcmToImage(f) $\equiv$ new ImageData($f.x$, $f.y$, stringToSequence($f.s$))
stringToSequence: string $\rightarrow$ sequence of $\mathbb{N}$
stringToSequence(str) $\equiv$ || (pv: string | $pv \in$ dcmFrame $\cdot$ String.parseToNum(pv))
pv is a string containing grayscale value for one pixel.

9 MIS of Image Data Structure

9.1 Generic Template Module

ImageData

9.2 Uses

none

9.3 Syntax

9.3.1 Exported Constants

9.3.2 Exported Types

ImageData = ?

9.3.3 Exported Access Programs

Name	In	Out	Exceptions
new ImageData	x : $\mathbb{N}$, y : $\mathbb{N}$, p: sequence of $\mathbb{N}$	imageData	badWidthInput, badHeightInput, badPixelValueLength
width	-	$\mathbb{N}$	
height	-	$\mathbb{N}$	
pixelValue	-	sequence of $\mathbb{N}$	

9.4 Semantics

9.4.1 State Variables

width: $\mathbb{N}$
height: $\mathbb{N}$
pixelValue: sequence

9.4.2 Environment Variables

none

9.4.3 Assumptions

imageData with 0 width or height is allowed. The software uses such imageData torepresent an empty image without successfully loading information from a frame. Image Verification Module 12 is able to identify it as not successfully loaded.

9.4.4 Access Routine Semantics

Input.width:

- output: *out* := width
- exception: none

Input.height:

- output: *out* := height
- exception: none

Input.pixelValue: [This access program is not given in the syntax.The state variable is missing. Is this a copy-paste error? —SS] [It's a copy-paste error. I changed it to the correct one —Author]

- output: *out* := pixelValue
- exception: none

new ImageData(x, y, p):

- transition: The parameters x and y are natural numbers, p is a sequence of natural numbers representing the pixel values from left to right, top to bottom. ImageData() is the conxtructor of this data structure, and it modifies the state variables using the following procedural specification:
  1. height := x
  2. height := y
  3. pixelValue := p
- output: := self
- exception: exc :=
  $x < 0 \implies$ badWidthInput
  $y < 0 \implies$ badHeightInput
  p.length $\neq x \times y \implies$ badPixelValueLength

9.4.5 Local Functions

None

10 MIS of Optimal Thresholds Calculation

10.1 Module

ThresCal

10.2 Uses

Constants 7, Input in Section 8, Image Data Structure 9

10.3 Syntax

10.3.1 Exported Constants

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
calculation	$j : \mathbb{N}$	-	skipCalculation, badResult
getUserChoice	$c : \mathbb{N}$	-	badChoiceInput
frameIndex	-	$\mathbb{N}$	
chosenThresNum	-	$\mathbb{N}$	
isThresValid	-	boolean	
k1	-	$\mathbb{N}$	
k2	-	$\mathbb{N}$	

10.4 Semantics

10.4.1 State Variables

frameIndex: $\mathbb{N}$
chosenThresNum: string
isThresValid: boolean
k1: $\mathbb{N}$
k2: $\mathbb{N}$
img: imageData

10.4.2 Environment Variables

None

10.4.3 Assumptions

None

10.4.4 Access Routine Semantics

ThresCal.frameIndex:

- output: *out* := frameIndex
- exception: none

ThresCal.chosenThresNum:

- output: *out* := chosenThresNum
- exception: none

ThresCal.isThresValid:

- output: *out* := isThresValid
- exception: none

ThresCal.k1:

- output: *out* := k1
- exception: none

ThresCal.k2:

- output: *out* := k2
- exception: none

calculation(j):
$j \in \mathbb{N}$ is the index of one imageData in the Input.loadedImages sequence.

- transition: [I don't think you need pseudo-code for this. A state based specification should be possible. —SS] [I deleted some pseudo-code and put more statements here —Author] This function modifies the state variables using the following procedural specification:

  1. frameIndex := j
  2. If ¬ Input.isLoaded[j], set isThresValid to False, and terminate this method.
  3. img := Input.loadedImages[j]
  4. getUserChoice()
  5. According to the chosen threshold numbers, calculate k1 (and k2):
     if chosenThresNum = 1, use the following formula to calculate k1,
     sigma2b1(k1)= $\max_{0<t1<255}$ sigma2b1($t1$);
     if chosenThresNum = 2, use the following formula to calculate k1 and k2,
     sigma2b2(k1,k2) = $\max_{0<t1<t2<255}$ sigma2b2($t1, t2$)).

6. According to the chosen threshold numbers, set isThresValid to True if thecalculated results follow the rules accordingly:
   $1 \leq k1 \leq 254$
   $1 \leq k1 < k2 \leq 254$

- output: none
- exception: $exc :=$
  $\neg$ Input.isLoaded[j] $\implies$ skipCalculation
  chosenThresNum $= 1 \wedge \neg(1 \leq k1 \leq 254) \implies$ badResult
  chosenThresNum $= 2 \wedge \neg(1 \leq k1 < k2 \leq 254) \implies$ badResult

getUserChoice(c):

- transition: The parameter c is a natural number representing user's choice. This function modifies the state variables using the following procedural specification:
  1. While chosenThresNum $\notin$ Constants.numsThres, repeat the following two steps, until get proper choice input from the user.
  2. Use hardware to display a message, asking for user's choice from Constants.numsThres for number of thresholds.
  3. If $c \in$ Constants.numsThres, chosenThresNum $:= c$
- output: none
- exception: $exc := c \notin$ Constants.numsThres $\implies$ badChoiceInput

10.4.5 Local Functions

n: $\mathbb{N} \to \mathbb{N}$
n(i) $\equiv +(pv.\mathbb{N}|pv \in$ img.pixelValue $\cdot(pv = i \implies 1|$ True $\implies 0))$
p: $\mathbb{N} \to \mathbb{R}$
p(i) $\equiv n(i)/($img.width $\times$ img.height$)$
prb: $\mathbb{N} \times \mathbb{N} \to \mathbb{R}$
prb($start, end$) $\equiv +(i.\mathbb{N}|i \in [start, end] \cdot p(i))$
m: $\mathbb{N} \times \mathbb{N} \to \mathbb{R}$
m($start, end$) $\equiv (+(i.\mathbb{N}|i \in [start, end] \cdot i \times p(i)))/prb(start, end)$
mg() $\equiv +(i.\mathbb{N}|i \in [0, 255] \cdot i \times p(i))$
sigma2b1: $\mathbb{N} \to \mathbb{R}$
sigma2b1($t1$) $\equiv prb(0, t1) \times (m(0, t1) - mg())^2 + prb(t1 + 1, 255) \times (m(t1 + 1, 255) - mg())^2$
sigma2b2: $\mathbb{N} \times \mathbb{N} \to \mathbb{R}$
sigma2b2($t1, t2$) $\equiv prb(0, t1) \times (m(0, t1) - mg())^2 + prb(t1 + 1, t2) \times (m(t1 + 1, t2) - mg())^2 +$
$prb(t2 + 1, 255) \times (m(t2 + 1, 255) - mg())^2$
[You didn't mention I should change here. But inspired by implementation, I removed one parameter from every function, and got rid of 4 functions by reuse existing ones. I think it's much better than before. —Author]

11 MIS of Output Module

11.1 Module

Output

11.2 Uses

ImageData 9, ThresCal in Section 10, ImageVerify in Section 12

11.3 Syntax

11.3.1 Exported Constants

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
displayThresholds	-	-	
writeOutput	s: string	-	skipSegmentation, badSegmentation, badDirectory
createSegmentation	-	-	
segImage	-	imageData	

11.4 Semantics

11.4.1 State Variables

img: imageData
segImage: imageData
j: $\mathbb{N}$

11.4.2 Environment Variables

outputFile: a bitmap file

11.4.3 Assumptions

If this module is intended to be used on frame j, than it should be called after using ThresCal in Section 10 on frame j. The reason isthat this module uses ThresCal.frameIndex to get current frame number.

11.4.4 Access Routine Semantics

Output.segImage:

- output: *out* := segImage

- exception: none

displayThresholds():

- transition: This function has the following procedural specification:

  1. If ThresCal.isThresValid(), continue.
  2. j := ThresCal.frameIndex
  3. If ThresCal.chosenThresNum == 1, use Hardware-Hiding Module to display the-following message:
     "The single threshold value for frame j is k= " + k1 + "."
  4. If ThresCal.chosenThresNum == 2, use Hardware-Hiding Module to display the-following message:
     ""The multiple threshold values for frame j are k1= " + k1 + ", k2= " + k2 + "."

- output: none

- exception: none

writeOutput(s): This method use segImage to write a outputFile to the environment using the following procedural specification:

The first step is not necessary for the Control Module in this document, which calls displayThresholds() that updates j, but it is necessary if displayThresholds() is not called in advance.

1. j := ThresCal.frameIndex
2. If ThresCal.isThresValid, continue.
3. createSegmentation()
4. if ImageVerify.verifyImageData(segImage) and ImageVerify.compareSizes(img,segImage), continue.
5. Use data from segImage to write an outputFile to the environment

- transition: none

- output: none

- exception: exc :=
  $\neg$ ThresCal.isThresValid $\implies$ skipSegmentation
  $\neg$ (ImageVerify.verifyImageData(segImage) $\wedge$ ImageVerify.compareSizes(img, segImage)) $\implies$ badSegmentation
  fail to write a file to the output directory $\implies$ badDirectory

createSegmentation():

- transition: This function modifies the state variables using the following procedural specification:

  1. img := Input.loadedImages[j]
  2. Use local references $c, k1, k2$ for state variables in ThresCal.
     - c = ThresCal.chosenThresNum
     - k1 = ThresCal.ThresCal.k1
     - k2 = ThresCal.ThresCal.k2
  3. pixelValue := $||(pv : \mathbb{N}|pv \in$ img.pixelValue $\cdot(c = 1 \implies (pv >$k1 $\implies 255|$ True $\implies 0)|\ c = 2 \implies (pv > k2 \implies 255|$ k2$\geq pv >$k1 $\implies 128|$ True $\implies 0))$
  4. segImage := new ImageData(img.width,img.height,pixelValue)

- output: none
- exception: none

11.4.5 Local Functions

None

12 MIS of Image Verification

12.1 Module

ImageVerify

12.2 Uses

none

12.3 Syntax

12.3.1 Exported Constants

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
verifyImageData	*img*: imageData	boolean	badSize, badPixelData
compareSizes	*img*1: imageData, *img*2: imageData	boolean	badSize2

12.4 Semantics

12.4.1 State Variables

None

12.4.2 Environment Variables

None

12.4.3 Assumptions

compareSizes(*img*1, *img*2) does not check if these two inputs are valid, it assumes that during the previous steps, the software has called verifyImageData(*img*) to verify these two inputs individually. [This accessprogram seems to be misnamed. The images are only being compared to verify that they are the same size. I assumed that returning True would mean that the images were the same. —SS] [I agree. Changed the names. —Author]

12.4.4 Access Routine Semantics

verifyImageData(*img*): The parameter *img* is an instance of Image Data Structure.

- transition: none
- output: := img.width $> 0 \wedge img$.height $> 0 \wedge (\forall pv \in img.\text{pixelValue} \cdot 0 \leq pv \leq 255)$

- exception: $exc :=$
  $img.\text{width} < 0 \vee img.\text{height} < 0 \implies \text{badSize}$
  $(\exists pv \in img.\text{pixelValue} \cdot\ pv < 0 \vee pv > 255) \implies \text{badPixelData}$

compareSizes($img1, img2$): The parameters $img1$ and $img2$ are instances of Image Data Structure.

- transition: none
- output: $:= (img1.\text{width} = img2.\text{width}) \wedge (img1.\text{height} = img2.\text{height})$
- exception: $exc :=$
  $img1.\text{width} \neq img2.\text{width} \vee img1.\text{height} \neq img2.\text{height} \implies \text{badSize2}$

12.4.5 Local Functions

None

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 https://pdfs.semanticscholar.org/2d2f/609de3c6d694b88b5b987b05bd5ec53be372.pdf.

13 Appendix

Table 2: Possible Exceptions

Message ID	**Error and Warning Message**
badInputFileName	Error: cannot find the file s.
badInputFileFormat	Error: the format of file s is not supported.
emptyFrameJ	Warning: frame j is not loaded.
badWidthInput	Error: image width cannot be negative.
badHeightInput	Error: image height cannot be negative.
badPixelValueLength	Error: the length of image pixel value sequence must equalto width $\times$ height.
skipCalculation	Warning: thresholds for frame j are not calculated.
badResult	Error: incorrect thresholds calculation for frame j
badChoiceInput	Error: input is not a number from the set, please read the following instructions carefully and try again:
skipSegmentation	Warning: frame j is not segmented nor saved.
badSegmentation	Error: frame j is not correctly segmented."
badDirectory	Error: Cannot write the output image to directory s
badSize	Error: invalid image size.
badPixelData	Error: One or more pixel values are outside the bound of [0, 255].
badSize2	Error: inconsistent image sizes.