Module Interface Specification for SPDFM

Author Name

November 16, 2020

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

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 SPDFM program. SPDFM is a software for simulating surface plasmon enhanced electric field and current density in meshed geometry.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at SPDFM repository on github.

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

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

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 (-∞, ∞)
imaginary	$\mathbb{I}$	any number of form $i \times \mathbb{R}$ where i is $\sqrt{-1}$

The specification of SPDFM 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, SPDFM 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	SPDFM Control Module Specification Parameters Module Input Parameters Modules Mesh Input Module SPD Calculations Control Module Output Module
Software Decision Module	Frequency Domain PDE Solver Module Data Structure Module

Table 1: Module Hierarchy

6 MIS of SPDFM Control Module

6.1 Module

main

6.2 Uses

- Data Structure
- Input Modules
  Input Parameters Module
  Mesh Input Module
- SPD Calculations Control Module
- Output Module

6.3 Syntax

6.3.1 Exported Constants

None.

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
main	-	-	-

6.4 Semantics

SPDFM Control Module is design to control the process flow in the software.

6.4.1 State Variables

None

6.4.2 Environment Variables

None

6.4.3 Assumptions

6.4.4 Access Routine Semantics

main():

- transition:

  Initiate global data object: doing so provides an empty closet that let me store data in it at different modules.

  ParamLoad: calling this module initiates reading environmental variables.

  GmshInput: calling this module initiates loading the mesh geometry

  SPDsimulator: controls process flow in the PDE solver.

  SPDoutput: exports the final results of the simulation

- output: None

- exception: None

6.4.5 Local Functions

None

7 MIS of Input Parameter Module

7.1 Module

ParamLoad

7.2 Uses

- Specification Parameters Module
- Data Structure

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
ParamLoad	string	-	FileError
verifyPol	-	-	PolarizationValueError, PolarizationRangeError
verifyDir	-	-	DirectionValueError, DirectionNormalityError, LightOrthogonalityError
verifyWL	-	-	WavelengthValueError, WavelengthRangeError
verifyT	-	-	TimeRangeError, TimeStepValueError, TimeStepRangeError
verifyEps	-	-	EpsValueError
verifyMu	-	-	MuValueError
verifyGamma	-	-	GammaValueError, GammaRangeError
verifyPFreq	-	-	PFreqValueError, PFreRangeError

7.4 Semantics

7.4.1 State Variables

data: object

7.4.2 Environment Variables

ParamLSfile: A file containing sequence of strings that provides data related to the light source.
ParamMPfile: A file containing sequence of strings that provides data related to the materials properties.

7.4.3 Assumptions

- ParamLoad will be called before the values of any state variables will be accessed.
- The file contains the string equivalents of the numeric values for each input parameter in order, each on a new line. The order of the input data is the same as in the table in R1 of the SRS document.

7.4.4 Access Routine Semantics

Function to load, verify, and store input data (R1 and R2 from SRS).

ParamLoad(pathLS, pathMP):

Still not clear how to get pathLS and PathMP

- transition: pathLS (light source data) and pathMP (material properties) are the file paths for the input files(InputParamFiles). The following procedure is performed:
  - Verify the format of the files to be .txt.
  - From ParamLSfile (located at pathLS), p (polarization of the incident light, $\mathbb{R}^3$ vector), d (direction of the incident light, $\mathbb{R}^3$ vector), wl (wavelength of the source), t (illumination time length, $\mathbb{R}$), and nst (number of time steps, $\mathbb{N}$) are extracted.
  - verifyPol
  - verifyDir
  - verifyWL
  - verifyT
  - Store p, d, wl, t, nst in the data structure as data.p, data.d, data.wl, data.t, and data.Nst.

–From ParamMSfile (located at pathMP), eps0 (environment permittivity, $\mathbb{R}$), mu0 (environment permeability, $\mathbb{R}$), gamma (plasmon damping term, $\mathbb{R}$), plasmonfreq (plasmon frequency of the medium, $\mathbb{R}$) are extracted.

– verifyEps

– verifyMu

– verifyGamma

– verifyPFreq

- output: None
- exception:

If the file addressed by pathLS or path MP doesn't exist	=> badFilePath
If the file format is not .txt	=> badFileFormat

7.4.5 Local Functions

verifyPol:

- output: None
- exception:

$(\exists p_i \in \ \mathbf{p} : \ p_i \notin \mathbb{R})$	=> PolarizationValueError
$\|p\| > p_{max}$ or $\|p\| < p_{min}$	=> PolarizationRangeError

verifyDir:

- output: None
- exception:

$(\exists d_i \ \in \ \mathbf{d} : \ d_i \notin \mathbb{R})$	=> DirectionValueError
$\|d\| \ \neq \ 1$	=> DirectionRangeError
$d.p! = 0$	=> LightOrthogonalityError

verifyWL:

- output: None

- exception:

$wl \notin \mathbb{R}$ => WavelenthValueError
$wl > wl_{max}$ or $wl < wl_{min}$ => WavelengthRangeError

verifyT:

- output: None
- exception:

$t \notin \mathbb{R}$ => TimeValueError
$t > t_{max}$ or $t < t_{min}$ => TimeRangeError
$nst \notin \mathbb{N}$ => TimeStepValueError
$\frac{t}{nst} > dt_{max}$ or $\frac{t}{nst} < dt_{min}$ => TimeStepRangeError

verifyEps:

- output: None
- exception:

$eps0 \notin \mathbb{R}$ => EpsValueError

verifyMu:

- output: None
- exception:

$Mu0 \notin \mathbb{R}$ => MuValueError

verifyGamma:

- output: None
- exception:

$t \notin \mathbb{R}$ => TimeValueError
$gamma > gamma_{max}$ or $gamma < gamma_{min}$ => GammaRangeError

verifyPFreq:

- output: None
- exception:

$plasmonfreq \notin \mathbb{R}$ => PFreqValueError

$plasmonfreq > plasmonfreq_{max}$ or $plasmonfreq < plasmonfreq_{min}$ => PFreqError

8 MIS of Specific Parameters Module

8.1 Module

SpecParam

8.2 Uses

N/A

8.3 Syntax

8.3.1 Exported Constants

From Table 2 in SRS

p_{min} := -10
p_{max} := 10
$t_{min} := 10^{-15}$
$t_{max} := 10^{-12}$
$dt_{min} := 10^{-15}$
$dt_{max} := 10^{-12}$
$R(\Omega)_{min} := 10^{-8}$
$R(\Omega)_{max} := 10^{-7}$

8.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
SpecParam	-	-	-

8.4 Semantics

N/A

9 MIS of Mesh Input Module

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

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

9.1 Module

GmshInput

9.2 Uses

- Data Structure

9.3 Syntax

9.3.1 Exported Constants

None.

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
GmshInput	string	-	FileError
MeshConvert	object	-	-

9.4 Semantics

9.4.1 State Variables

Data: object

9.4.2 Environment Variables

inputMesh: A .mesh file containing the data related to the meshed geometry.

9.4.3 Assumptions

None.

9.4.4 Access Routine Semantics

Function to load, convert and verify the mesh file (R1, R2)

Still not clear how to get pathMESH!

gmshInput(pathMESH):

- transition: pathMESH is the file path for the input mesh file. The following procedure is performed:
  - Verify the format of the file to be .mesh.
  - Load mesh object, GMESH, from the input file.
  - MeshConvert
  - Geometry is stored in the data structure as data.Mesh.
- output: None.
- exception:

If the file addressed by pathMESH doesn't exist	=> badMeshFilePath
If the file format is not .mesh	=> badMeshFileFormat

9.4.5 Local Functions

MeshConvert(GMSH):

- transition:
  - load input mesh, GMSH.
  - convert mesh input format:

    data.XMesh = mesh.convert(GMSH)
- output: None.
- exception: None.

10 MIS of SPD Simulator Module

10.1 Module

SPDSimulator

10.2 Uses

- Frequency Domain PDE Solver Module

10.3 Syntax

10.3.1 Exported Constants

None.

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
SPDSimulator	-	-	-

10.4 Semantics

10.4.1 State Variables

Data: object

10.4.2 Environment Variables

N/A

10.4.3 Assumptions

None.

10.4.4 Access Routine Semantics

Functions to calculate the Electric field and Electric Current density in the give mesh and illumination (satisfies R3, R4). At this moment only Frequency domain is being solved SPDsimulator only calls for FreqSolver().

SPDSimulator():

- transition:

  FreqSolver()

- output: None.
- exception: None.

10.4.5 Local Functions

None.

11 MIS of Frequency Domain PDE Solver Module:

11.1 Module

FreqSolver

11.2 Uses

- Data Structure Modules

11.3 Syntax

11.3.1 Exported Constants

None.

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
FreqSolver	-	-	-

11.4 Semantics

11.4.1 State Variables

data: object

11.4.2 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]

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

FreqSolver():

– Load the inputs from the data object
  – Setup the Nedelec Ansatz function space for a single parameter:
  FS = FunctionSpace(data.XMesh,"N1curl", 2)

– Setup the space element. As shown in IM2 in the SRS the system of equations that needs to be solved here is a compound system of equations that has two unknown parameters electric field density and electric current density vectors. Each of these parameters are complex, therefore. each need to be split into imaginary and real parts:
element = MixedElement([FS, FS, FS, FS])
– Define the combined Function Space:
ComboV = FunctionSpace(mesh, element)
– define the the test function:
E_r^{test}, E_i^{test}, J_r^{test}, J_r^{test} = TestFunction(ComboV)
– define the the trial function:
U = Function(ComboV)
E_r^{trial}, E_i^{trial}, J_r^{trial}, J_r^{trial} = split(U)
– AtTheBoundary()
– Load the variational form of equation system: F = f(data.Matprop, BoundariC, U, ComboV)
– For all the frequencies in the domain,data.freq[k] (need to specify domain somewhere):
: Solve (F==0,U)
data.E_r[k], data.E_i[k], data.J_r[k], data.J_r = split(U)

11.4.5 Local Functions

AtTheBoundary():

- transition: sets the Dirichlet boundary condition in the infinity, Neumann in the surrounding, and scattering at the interface. Light source data is used here.

12 MIS of Data Structure Module

12.1 Module

data

12.2 Uses

- Hardware Hiding module

12.3 Syntax

12.3.1 Exported Constants

None.

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
store	string	-	-
load	string	object	-
StoreMesh	object	-	-
loadMesh	-	object	-

12.4 Semantics

12.4.1 State Variables

data:object

- data.p = $[p_x,\ p_y,\ p_z] \in \mathbb{R}^3$
- data.d = $[d_x,\ d_y,\ d_z] \in \mathbb{R}^3$
- data.wl = wl $\in \mathbb{R}$
- data.t = t $\in \mathbb{R}$
- data.Nst = nst $\in \mathbb{N}$
- data.eps0 = eps0 $\in \mathbb{R}$
- data.mu0 = mu0 $\in \mathbb{R}$
- data.beta = beta $\in \mathbb{R}$
- data.gamma = gamma $\in \mathbb{R}$

- data.pfreq = pfreq $\in \mathbb{R}$
- data.Xmesh = Mesh object
- data.BoundaryC = BC object
- data.freq = [list] $\in \mathbb{R}^{Nst}$
- data.E_i = [list] $\in \mathbb{R}^{Nst}$
- data.E_r = [list] $\in \mathbb{R}^{Nst}$
- data.J_i = [list] $\in \mathbb{R}^{Nst}$
- data.J_r = [list] $\in \mathbb{R}^{Nst}$

12.4.2 Environment Variables

N/A

12.4.3 Assumptions

None.

12.4.4 Access Routine Semantics

store(a,b):

- transition: data.a = b
- output:
- exception:

load(a):

- transition:
- output: data.a
- exception:

storeMatrix(a):

- transition: data.XMatrix = a
- output:
- exception:

LoadMatrix():

- transition:
- output: data.XMatrix
- exception:

12.4.5 Local Functions

None.

13 MIS of Output 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]

13.1 Module

Output

13.2 Uses

- Data Structure Module

13.3 Syntax

13.3.1 Exported Constants

None.

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
VtkSaver	-	Vtk	-
AmpOut	-	string	-
listOut	-	string	-

13.4 Semantics

13.4.1 State Variables

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

13.4.2 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]

13.4.3 Assumptions

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

13.4.4 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]

13.4.5 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.

14 Appendix

[Extra information if required —SS]