Module Interface Specification for SPDFM

S. Shayan Mousavi M.

December 2, 2020

1 Revision History

Date	Version	Notes
Nov 23, 2020	1.0	Notes

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/shmouses/SPDFM/blob/master/docs/SRS/SRS.pdf.

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 Input Parameter Module Constant Parameters Module Mesh Input Module SPD Simulator 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 (Section 12)
- Input Parameter Module (Section 7)
- Mesh Input Module (Section 9)
- SPD Simulator Control Module (Section 10)
- Output Module (Section 13)

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. With respect to the SRS document, current module organizes all other modules to satisfy all the requirements that specified in SRS. This module also help maintainability and expandability of SPDFM by classifying different parts of the code.

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: Control the flow input data, calculation, and the output data by following below steps:

  what module is this in?

  Initiate global data object: doing so provides an empty framework that lets user store data at different modules.

  ParamLoad: This module inputs all the parameters and stores them in the data structure.

  MshInput: This module loads and prepares the mesh geometry for finite element calculations.

  SPDsimulator: Controls process flow and interactions with the FEniCS PDE solver. FEniCS PDE solver is a library used in this study for solving non-local hydrodynamic PDEs (see IM 2 in the SRS document). Interested readers can find more information about FEniCS at Alnæs et al. (2015); Logg et al. (2012).

  SPDoutput: Exports the final results of the simulation into .vtk file.

- 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 (Section 8)
- Data Structure (Section 12)

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
ParamLoad	string	-	FileError
VrifyParam	-	-	PolarizationRangeError, DirectionNormalityError, LightOrthogonalityError, WavelengthRangeError, GammaRangeError

Do you want to expose VrifyParam in the interface? Will you ever use VrifyParam distinct from using ParamLoad? If not, you could make VrifyLoad a local function, used by ParamLoad

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 as below:

  # Data in ParamLSfile:

  line 1: p_x, p_y, p_z

  line 2: d_x, d_y, d_z

  line 3: WL_{min}

  line 4: WL_{max}

  line 5: NumLS

  # Data in ParamMPfile:

  line 1: eps0

  line 2: mu0

  line 3: gamma

  line 4: PlasmaFreq

  line 5: Beta

7.4.4 Access Routine Semantics

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

ParamLoad(pathLS, pathMP):

- transition: pathLS (light source data) and pathMP (material properties) are the file paths for the input files which user provides. The following procedure is performed:
  - Verify the format of the files to be .csv.

  - From ParamLSfile (located at pathLS) below parameters are obtained (As PDE equation (IM 2 SRS document) is being solve in the frequency domain light source should have a minimum and a maximum wavelength to specify the interval of the frequency domain. In this regard, number of frequencies in the intervals should be input as NumLS):

    Polarization of the incident light($\mathbb{R}^3$ vector): data.pol := p := [p_x, p_y, p_z]

    Direction of the incident light ($\mathbb{R}^3$ vector): data.dir := d := [d_x, d_y, d_z]

    Minimum wavelength of the light source ($\mathbb{R}$): data.WLmin := WL_{min}

    Maximum wavelength of the light source ($\mathbb{R}$): data.WLmax := WL_{max}

    Number of different wavelengths in the interval ($\mathbb{N}$): data.NumLS := NumLS

– verifyPol

– verifyDir

– verifyWL

–From ParamMSfile below information is obtained:

Environment permittivity ($\mathbb{R}$): data.eps0 := eps0

Environment permeability ($\mathbb{R}$): data.mu0 := mu0

Plamson damping parameter ($\mathbb{R}$): data.damp := gamma

Plasma frequency ($\mathbb{R}$): data.Pfreq := PlasmaFreq

Fermi velocity proportionality ($\mathbb{R}$): data.beta := Beta

– verifyGamma

- output: None
- exception: exc :=

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

7.4.5 Local Functions

verifyPol:

- output: None
- exception: exc :=

  $\neg(data.PminLmt < \|p\| < data.PmaxLmt)$ => PolarizationRangeError

verifyDir:

- output: None
- exception: exc :=

  $\|d\| \neq 1$ => DirectionRangeError
  $d.p! = 0$ => LightOrthogonalityError

verifyWL:

- output: None
- exception: exc :=

  $\neg(data.WLminLmt < WL_{min} < WL_{max} < data.WLmaxLmt)$ =>WavelengthRangeError

verifyGamma:

- output: None
- exception: exc :=

  $\neg(data.dampMinLmt < gamma < data.dampMaxLmt)$ => GammaRangeError

8 MIS of Constant Parameters Module

8.1 Module

ConstParam

8.2 Uses

- Data Structure (Section 12)

8.3 Syntax

8.3.1 Exported Constants

From Table 2 in SRS update the values in the data object:

data.PminLmt := -10

data.PmaxLmt := 10

data.RadiusMinLmt := 10

data.RadiusMaxLmt := 100

data.WLminLmt := 200

data.WLmaxLmt := 1000

data.dampMinLmt:= 0.01

data.dampMaxLmt:= 1

Although currently only constants in the SPDFM are exported from specification parameters table in SRS document (Table 2), this module also considers future expansions of the program. In this regard, more constants can be added to this module in the future if needed.

This looks like you are doing a state transition for the project data. abstract

If that is your intention, you should document it that way. Is the plan really to have constants in this module, then you shouldn't connect them to data at all not within this module

This information should go in the Considerations section.

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
ConstParam	-	-	-

8.4 Semantics

N/A

9 MIS of Mesh Input Module

9.1 Module

GmshInput

9.2 Uses

- Data Structure (Section 12)

9.3 Syntax

This is very abstract. Can you give the class name of the objects?

9.3.1 Exported Constants

None.

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
GmshInput	string		FileError
MeshConvert	object	object	-

9.4 Semantics

9.4.1 State Variables

For inputting the mesh object in this module, meshio toolbox is used. Interested readers for better understanding properties of meshio mesh object are referred to Schlömer et al. (2019) and Dai (2017).

Data: mesh object

9.4.2 Environment Variables

inputMesh: A .msh 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. This module satisfies R1 and R2 requirements from the SRS document.

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 .msh.
  - Load mesh object, GMESH, from the input file.
  - MeshConvert
  - Geometry is stored in the data structure: data.XDMFmesh := XDMFmesh
- output: None
- exception: exc :=

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

MeshConvert(GmeshFile):

- transition: None
- output: XDMFmesh which is converted GmeshFile (.mesh format) into .xdmf format (which is an acceptable format for FEniCS to initiate FEM solver)
- exception: None

9.4.5 Local Functions

None

10 MIS of SPD Simulator Module

10.1 Module

SPDSimulator

10.2 Uses

- Frequency Domain PDE Solver Module (Section 11)

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 given mesh and illumination (satisfies R3, R4 from SRS document). In fact, this module will control flow and order of the different PDE solver units. Although at the moment it seems like this module only calls "Frequency Domain PDE Solver Module (Section 11)", it gives expandability to the software to have more PDE solver modules in the future. This module is aligned with the second non-functional requirement (NR2) from the SRS document which is maintainability and expandability of the software.

SPDSimulator():

- transition:
  - FreqSolver()
- output: None.
- exception: None.

10.4.5 Local Functions

None

what state variable is modified? How is it changed?

A natural lang. spec. is fine, if it is not difficult to formalize

If all the module does is call FreqSolver(), do you need this module?

11 MIS of Frequency Domain PDE Solver Module:

11.1 Module

FreqSolver

11.2 Uses

- Data Structure Modules (Section 12)

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

None

11.4.3 Assumptions

None

11.4.4 Access Routine Semantics

This module specifically feeds the frequency domain PDE equations to the FEniCS finite element PDE solver. FEniCS toolbox is used for all finite element setups in SPDFM. To better understand the FEniCS implementation and data structure, readers are encouraged to look up Alnæs et al. (2015) and Logg et al. (2012). The FEniCS-related implementations which include defining function space, element space, trial function, test function, and inputting variational form of the system of equations are not discussed in this document as these descriptions are part of FEniCS toolbox and are discussed in details in references mentioned above.

FreqSolver():

- transition: FEniCS toolbox will receive nonlocal hydrodynamic equation system (IM2 SRS document) to calculate the electric field and electric current at different given frequencies and the mesh.

  – Setting up FEniCS environment.

  – importing below equations to the FEniCS, all the parameters in the equation are defined in IM2 and also the table of symbols in the SRS document:

$$-\int_\Omega \beta^2(\nabla.\psi)(\nabla.\mathbf{J}_{HD})dV + \omega(\omega+i\gamma)\int_\Omega \psi.\mathbf{J}_H DdV - \ i\omega\omega_p^2\int_\Omega \psi.\varepsilon_0\mathbf{E}dV = 0$$

$$\int_\Omega((\nabla\times\phi).(\mu_0^{-1}\nabla\times\mathbf{E}) - \omega^2\phi.\epsilon_{local}\mathbf{E})dV + \int_{\partial\Omega}\phi.(\mathbf{n}\times(\mu_0^{-1}\nabla\times\mathbf{E}))dA = i\omega\int_\Omega\phi.\mathbf{J}_{HD}dV$$

$$Boundary\ Conditions:$$

$$\int_\Omega((\nabla\times\phi).(\mu_0^{-1}\nabla\times\mathbf{E}) - \omega^2\phi\epsilon_{local}\mathbf{E}_i)dV + \int_{\partial\Omega}\phi.DtN(\mathbf{E})dA \ \ - i\omega\int_\Omega\phi.\mathbf{J}_{HD}dV = -\int_{\partial\Omega}\phi.(n\times(\mu_0^{-1}\nabla\times\mathbf{E}_i))dA + \int_{\partial\Omega}\phi.DtN(\mathbf{E}_i)dA$$

$$n.\mathbf{J}_{HD} = 0 \ \ on\ \partial\Omega$$

For importing the data to FEniCS, as complex numbers are not defined in this toolbox parameters and equations should be separated into real and imaginary parts.

$$\mathbf{J}_{HD} = \mathbf{J}_{HD}^{real} + i\mathbf{J}_{HD}^{img}$$
$$\mathbf{E} = \mathbf{E}^{real} + i\mathbf{E}^{img}$$

–Store the result for each frequency in the Data.FDres.

$$data.FDres = \begin{pmatrix}\mathbf{E}^{real}\\ \mathbf{E}^{img}\\ \mathbf{J}_{HD}^{real}\\ \mathbf{J}_{HD}^{img}\end{pmatrix}$$

12 MIS of Data Structure Module

12.1 Module

data

Is this module an abstract object (there is only one instance), or is it an abstract data type (potentially multiple instances)?

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
init	-	-	-

12.4 Semantics

12.4.1 State Variables

data:object

- data.pol $\in \mathbb{R}^3$
- data.dir $\in \mathbb{R}^3$
- data.PminLmt $\in \mathbb{R}$
- data.PmaxLmt $\in \mathbb{R}$
- data.WLmin $\in \mathbb{R}$
- data.WLmax $\in \mathbb{R}$
- data.WLminLmt $\in \mathbb{R}$
- data.WLmaxLmt $\in \mathbb{R}$
- data.NumLS $\in \mathbb{N}$
- data.eps0 $\in \mathbb{R}$
- data.mu0 $\in \mathbb{R}$

- data.beta $\in \mathbb{R}$
- data.damp $\in \mathbb{R}$
- data.dampMinLmt $\in \mathbb{R}$
- data.dampMaxLmt $\in \mathbb{R}$
- data.Pfreq $\in \mathbb{R}$
- data.XDFMmesh : Mesh object
- data.FreqRes $\in \mathbb{R}^4$

12.4.2 Environment Variables

N/A

12.4.3 Assumptions

None

12.4.4 Access Routine Semantics

N/A

12.4.5 Local Functions

None

13 MIS of Output Module

13.1 Module

Output

13.2 Uses

- Hardware Hiding Module
- Data Structure Module (Section 12)

13.3 Syntax

13.3.1 Exported Constants

None.

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
Output	-	-	-
vtkExport	-	string (.vtk file)	-

13.4 Semantics

13.4.1 State Variables

None

13.4.2 Environment Variables

None

13.4.3 Assumptions

None

13.4.4 Access Routine Semantics

Output():

- transition: Control different output units. The current version of the code only uses one unit as below:
  - vtksaver

- output: None
- exception: None

vtkSaver():

- transition: None
- output: Exported vtkFile:

  vtkFile := vtk export of data.FreqRes
- exception: None

13.4.5 Local Functions

None

References

Martin Alnæs, Jan Blechta, Johan Hake, August Johansson, Benjamin Kehlet, Anders Logg, Chris Richardson, Johannes Ring, Marie E Rognes, and Garth N Wells. The fenics project version 1.5. *Archive of Numerical Software*, 3(100), 2015.

William W Dai. High performance io tools for mesh in numerical simulations. In *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1582–1587. IEEE, 2017.

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.

Anders Logg, Kent-Andre Mardal, and Garth N Wells. Finite element assembly. In *Automated Solution of Differential Equations by the Finite Element Method*, pages 141–146. Springer, 2012.

N Schlömer et al. Meshio, 2019.