2018

Orhan Eroglu, Dylan R. Boyd, Mehmet Kurum

Infor**M**ation **PR**oc**ES**sing and **S**ensing (IMPRESS) Lab

http://impress.ece.msstate.edu/

SCoBi Simulator Developer's Manual

v1.0

Table of Contents

1. Introduction

1.1. General

SCoBi, the **S**ignals of Opportunity (SoOp) **Co**herent **Bi**static scattering model and simulator, is a framework that is designed to use the Information Processing and Sensing (IMPRESS) Lab's fully coherent scattering model within a user-friendly simulation interface to enable comprehensive analysis of bistatic SoOp configurations for land applications. The current SCoBi release (v1.0.0) boasts the following capabilities:

- Fully polarimetric analysis with any combination of linear and/or circular polarizations
- Antenna property realizations including antenna orientation, pattern, and cross-polarization coupling
- Interferometric effect implementation caused by complex voltage and beamforming
- Geometry effects induced by altitude, orientation, and spreading loss over vegetation depth and soil moisture profile

SCoBi generates power and complex field outputs for the direct signals between the transmitter and the receiver, and the coherent reflection coefficient and reflectivity outputs regarding the specular point between the antennas. The SCoBi model is capable of handling the diffuse vegetation scattering mechanisms through Monte Carlo simulations via distorted Born approximation, but this feature is not included in the current version of the SCoBi simulator framework (due to the dominance of the specular term over diffuse contribution [1], [2]). A comprehensive description of the theory behind the model can be found in [1].

1.2. System Requirements

SCoBi supports the following platforms and environments:

- OS: Windows 10 64 bit
- Environment: MATLAB R2015a (the oldest version that is tested with SCoBi) or above

1.3. Downloading and Installation

SCoBi software can be accessed from the following github repository:

https://github.com/impresslab/SCoBi

It can also be downloaded from the following URL:

http://impress.ece.msstate.edu/impress-lab/software/scobi/source-code/

There is no installation requirement for the current version. In other words, it can be directly run from within the source code when it is downloaded.

1.4. About This Document

The SCoBi Developer;s Manual has been prepared to document the architectural design of the SCoBi simulator framework, to help the potential developers understand the implementation

details, and to expedite further extensions to the system. This document has its own version number convention regardless of that of the SCoBi simulator software. The two-digit version number of this document represents the major updates (to the document) in the first digit and minor changes in the second digit. On the other hand, the three-digit version number of the SCoBi software represents the major updates to the framework in the first digit, minor changes in the second digit, and bug-fixes in the third digit.

1.5. Help SCoBi Improve

Please send us an email via the following address to make requests or to report any bugs through using the software:

impress@ece.msstate.edu

1.6. How to Cite This Study?

The SCoBi software is open-source under GNU General Public License (GPL) and freely available with its documentation, design, and tutorial videos. However, the developers of the SCoBi model and the simulator would appreciate those who cite the corresponding studies below in the case they are used:

SCoBi Model:

M. Kurum, M. Deshpande, A. T. Joseph, P. E. O'Neill, R. Lang, and O. Eroglu, "SCoBi-Veg: A generalized bistatic scattering model of reflectometry from vegetation for Signals of Opportunity applications," *IEEE Trans. Geosci. Remote Sensing, Press*.

SCoBi Simulator:

O. Eroglu, Dylan R. Boyd, and M. Kurum, "SCoBi: A free, open-source, SoOp coherent bistatic scattering simulator framework," IEEE Geosci. and Remote Sensing Magazine, Review.

2. SCoBi Overall Architecture

The SCoBi simulator framework has been developed by an iterative and incremental development process (including requirements analysis, design, implementation, testing, and deployment). For instance, the requirements analysis was mostly done during the creation of the SCoBi model [1] in several years, and then the simulator framework has been designed, implemented, and tested with increments. However, many iterations have been performed on the requirements, design, and implementation after results of the tests and findings of the studies [1], [2] accomplished by the preliminary SCoBi versions. Our tests have demonstrated the verification of the SCoBi simulator framework. In other words, tests show that the simulator meets the requirements of the SCoBi model. The validation of SCoBi is mature to some degree since the simulated results in [1], [2] satisfy real-world expectations. Moreover, airborne data will be used for experimental validation in the future. For this purpose, observatory data are currently being collected over several terrains by using a drone. The maintenance of the SCoBi product will be handled by the authors for scheduled improvements (such as improving exception-handling mechanisms or adding new SoOp analysis types), user requests, and possible scientific collaborations.

The SCoBi source code has been implemented in the MATLAB R2017a environment; however, it is compatible with the versions above MATLAB R2015a (The oldest version which SCOBi was tested with) within MS Windows Operating System (Windows 10 64-bit). SCoBi does not require additional toolboxes or plugins of the MATLAB environment. MATLAB has been chosen for the development because of its common use among the researchers, efficient handling of the matrices, simple scripting features, and plotting capabilities. Both the structural and behavioral design models of the SCoBi software can be found in the Sparx Systems' Enterprise Architect design file [3]. Enterprise Architect application requires a purchased license to use; however, the structural and behavioral designs are described in detail in this document as well.

2.1. Structural Design

The SCoBi architectural design is mainly achieved with the procedural programming (PP) principles that MATLAB intrinsically supports. However, object oriented programming (OOP) design and implementation principles are also utilized as needed for advanced design, data encapsulation, manipulation, code organization and readability, and maintenance purposes. Combining two design approaches is for the purpose of having the enumerated advantages of the OOP design while exploiting MATLAB's procedural scripting capabilities. For instance, the simulation engine (***runSCoBi.m***) is simply operated by a MATLAB procedure (function) implementation, whereas the dynamic and static system parameters are handled with the help of several classes with singleton pattern features. The software packages within the source code are determined with respect to the relational hierarchy between each software entity (MATLAB functions or classes). Each package consists of several functions and/or classes. The UML (Unified Modeling Language) package diagram for the SCoBi source code (***/source/lib/***) packages is shown in **Fig. 1**. The ***runSCoBi.m*** function is directly under the ***lib*** package. It uses several packages to perform specific tasks in order to compute the model's output; for instance, the ***gui*** package is used to obtain the user inputs for simulations, ***init*** and ***param*** packages are used to initialize the simulation parameters by using the information (inputs) from the ***gui*** package, and the ***main*** package is used to perform every simulation iteration. The ***main*** package

uses the ***param*** package to manipulate parameters-related tasks, the ***bistatic*** package to handle the bistatic geometry, the ***ground*** package to account for ground operations (dielectric calculation, and specular reflection), the ***multilayer*** and ***vegetation*** packages if involved in a simulation, and the ***products*** package to create and store simulation outputs. There are information flows from the ***bistatic***, ***ground***, ***multilayer*** (if included), and ***vegetation*** (if included) packages to the ***products*** package.

The SCoBi design file (created in the Enterprise Architect tool) also includes the UML class model of the software (***lib*** package). In fact, class models are dedicated to the class instances in the OOP designs; however, we employed this model to depict the entire structural relations (usage, information flow, or inheritance) between the source code entities (MATLAB functions and classes). Although this is not a valid use of the class model, it can help the developers understand the general structure of SCoBi. The class model of the entire SCoBi software is too large to show in this document, and it can only be viewed from within the design file. On the other, because the overall SCoBi class model is highly complicated, we also provide class models for the ***runSCoBi.m*** and ***mainSCoBi.m*** functions, which are the simulation engine and simulation iterator functions, respectively. These two models only show the dedicated function (***runSCoBi.m*** or ***mainSCoBi.m***) and its first-degree relations with the other source code entities. The class models for runSCoBi.m and mainSCoBi.m are shown in **Fig. 2** and **Fig. 3**, respectively.

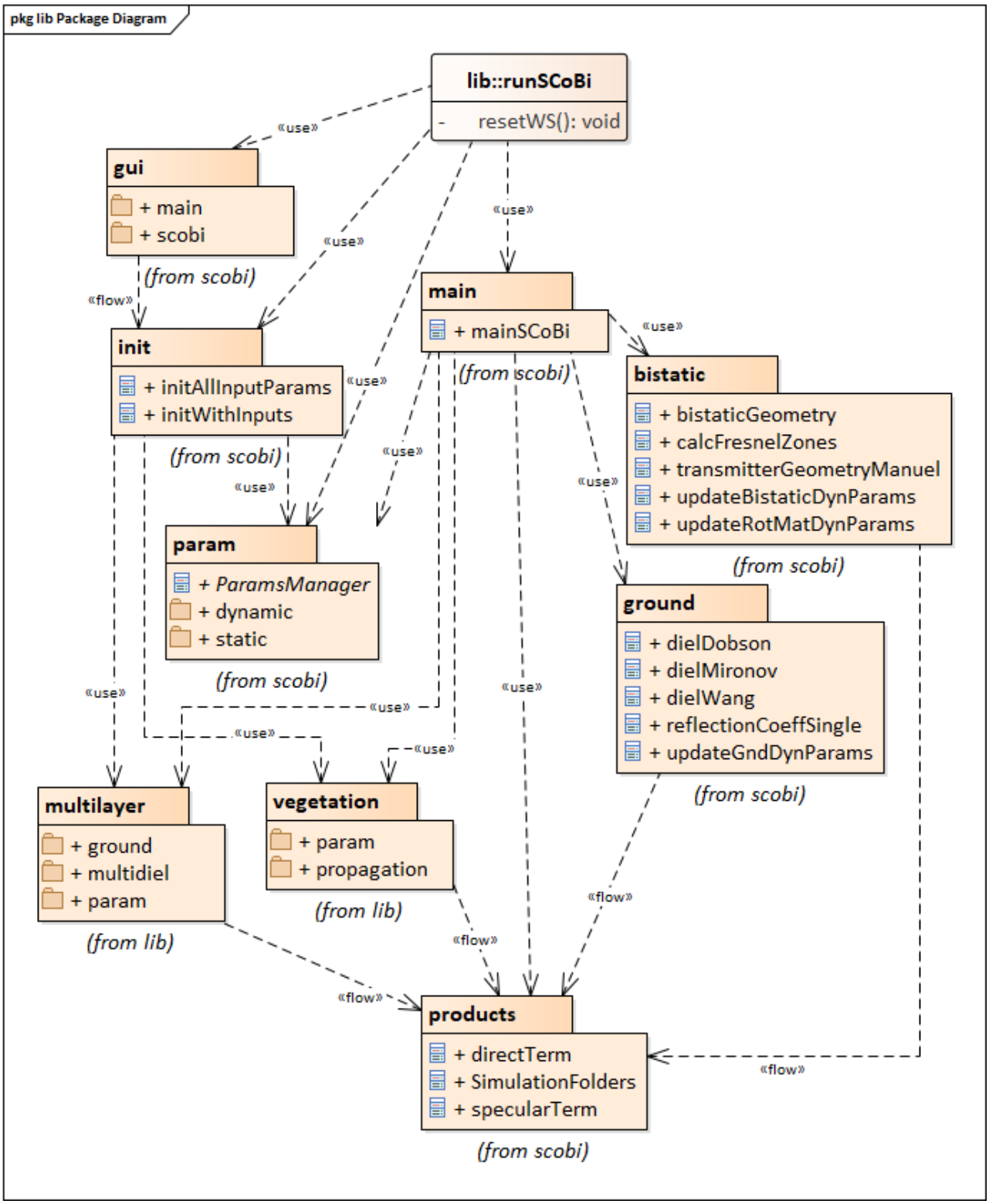

Fig. 1. SCoBi Package Diagram

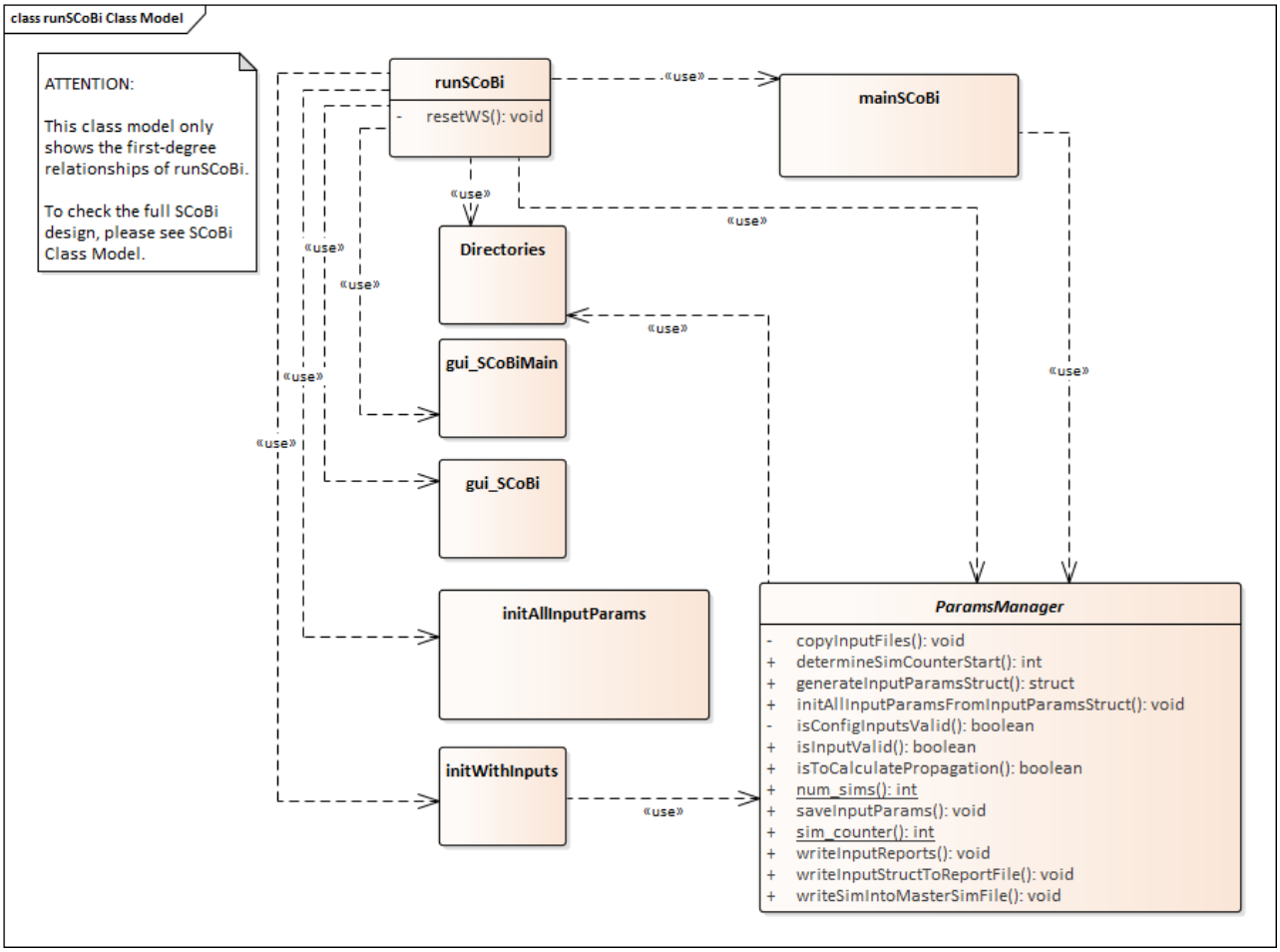

Fig. 2. runSCoBi.m Class Model. This model mimics actual OOP class models to show the structural hierarchy between functions and classes in SCoBi. It shows only the runSCoBi.m function and its first-degree relations.

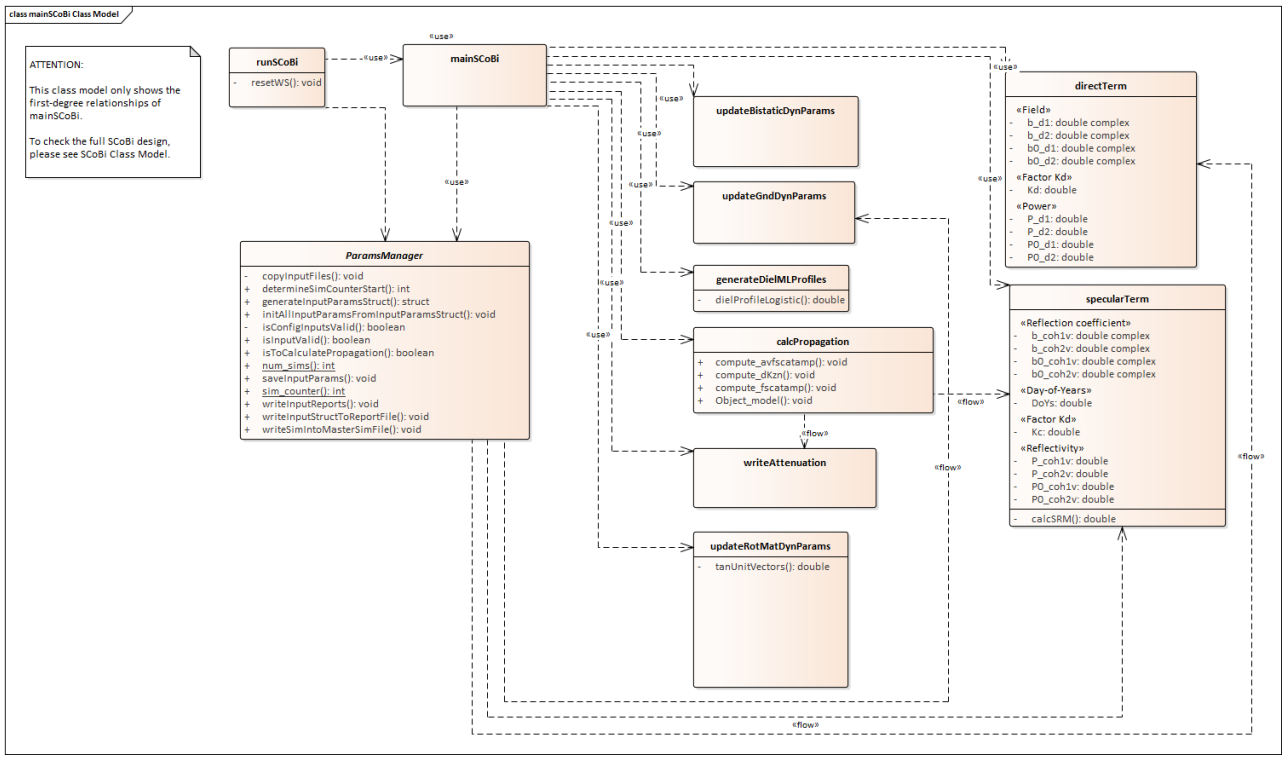

Fig. 3. mainSCoBi.m Class Model. This model mimics actual OOP class models to show the structural hierarchy between functions and classes in SCoBi. It shows only mainSCoBi.m and its first-degree relations.

2.2. Behavioral Design

UML behavioral diagrams such as sequence and activity diagrams are specialized to demonstrate the dynamic aspects of software programs. We used the UML activity diagrams to show the temporal flow of the SCoBi simulator. The SCoBi design file contains two activity diagrams for both the simulation engine and simulation iterator since these two deals with the overall flow. The SCoBi simulator framework always starts with running the ***runSCoBi.m*** function. The activities and the decisions that are performed within this function are shown in Fig. 9. In summary, it gets the user inputs, initializes the parameters and the simulation by using these inputs with the help of input validation controls, and calls the simulation iterator (***mainSCoBi.m***) after writing the simulation reports.

The ***runSCoBi.m*** determines the required number of simulation iterations for a chosen simulation by using the parameters manager class. This is because the simulation mode, *Snapshot* or *Time-series*, in conjunction with the system and configuration inputs may change the number of total simulation iterations. Details of this phenomenon are described in both the user's manual [4] and the tutorial videos [5]. The simulation engine then runs the simulation iteration function (***mainSCoBi.m***) for the number of simulations.

The simulation iteration function, ***mainSCoBi.m***, is always the same regardless of the analysis type. However, the analysis type and input selections affect the package usage and procedure calls within ***mainSCoBi.m***. For instance, a vegetation analysis requires SCoBi to use the software package, vegetation, and call several related procedures such as propagation calculation.

The flow of the ***mainSCoBi.m*** function is shown in a UML activity diagram in . Since this function is called in every iteration of the simulation engine, it handles the updates and calculations related to the iterative steps. For instance, it starts with updating the dynamic parameters for the bistatic geometry and ground surface. These updates are needed in each iteration since the simulation inputs may contain changing transmitter orientation angles and land geophysical parameters such as VSM and RMSH. If the Ground Structure is Multi-layered, the multi-layer dielectric profiles are computed. If the Ground Cover is Vegetation, the vegetation propagation is calculated. Furthermore, the vegetation attenuation values are written into an Excel file if chosen by the user. The polarization rotation matrices for the local antenna coordinate systems are also required to be updated due to the changes in the bistatic geometry in each iteration. Finally, the current iteration gets ready to generate the SoOp deliverables (the direct and specular contributions) for the parameters in the iteration, and stores the iteration calculations incrementally to the simulation outputs.

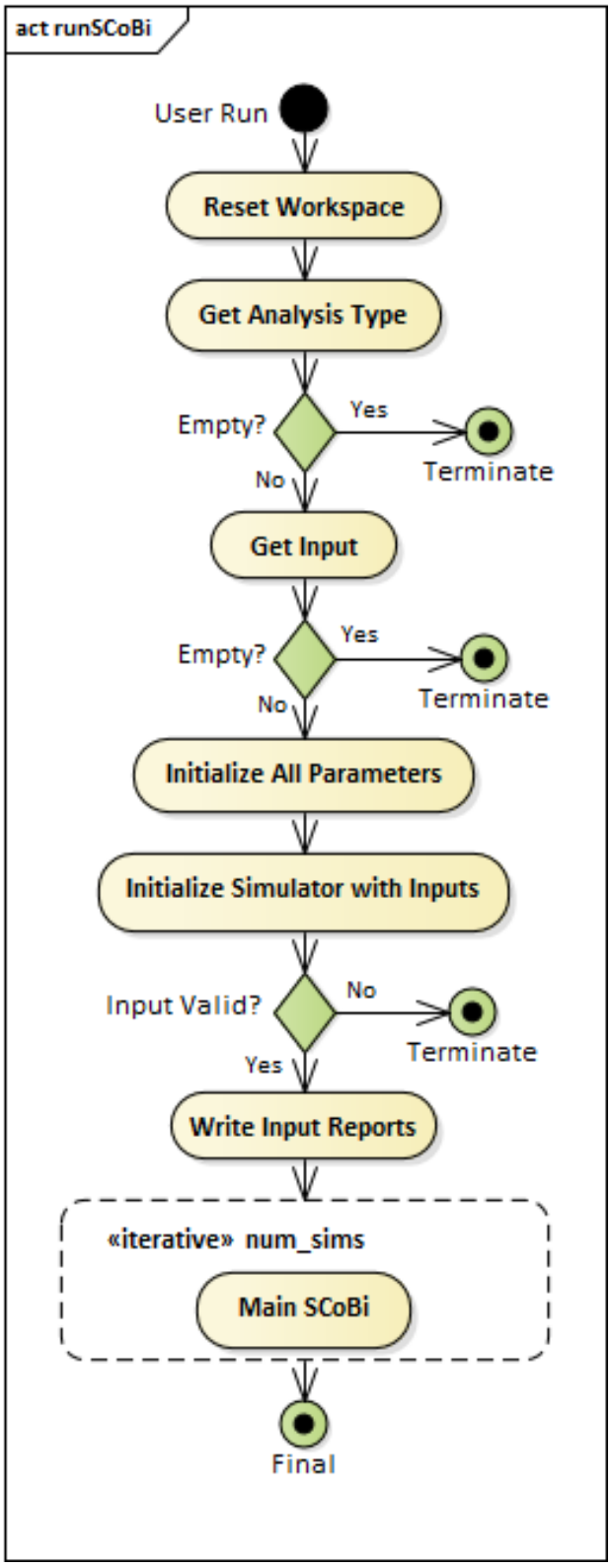

Fig. 4. runSCoBi.m (Simulation engine) Activitiy Diagram

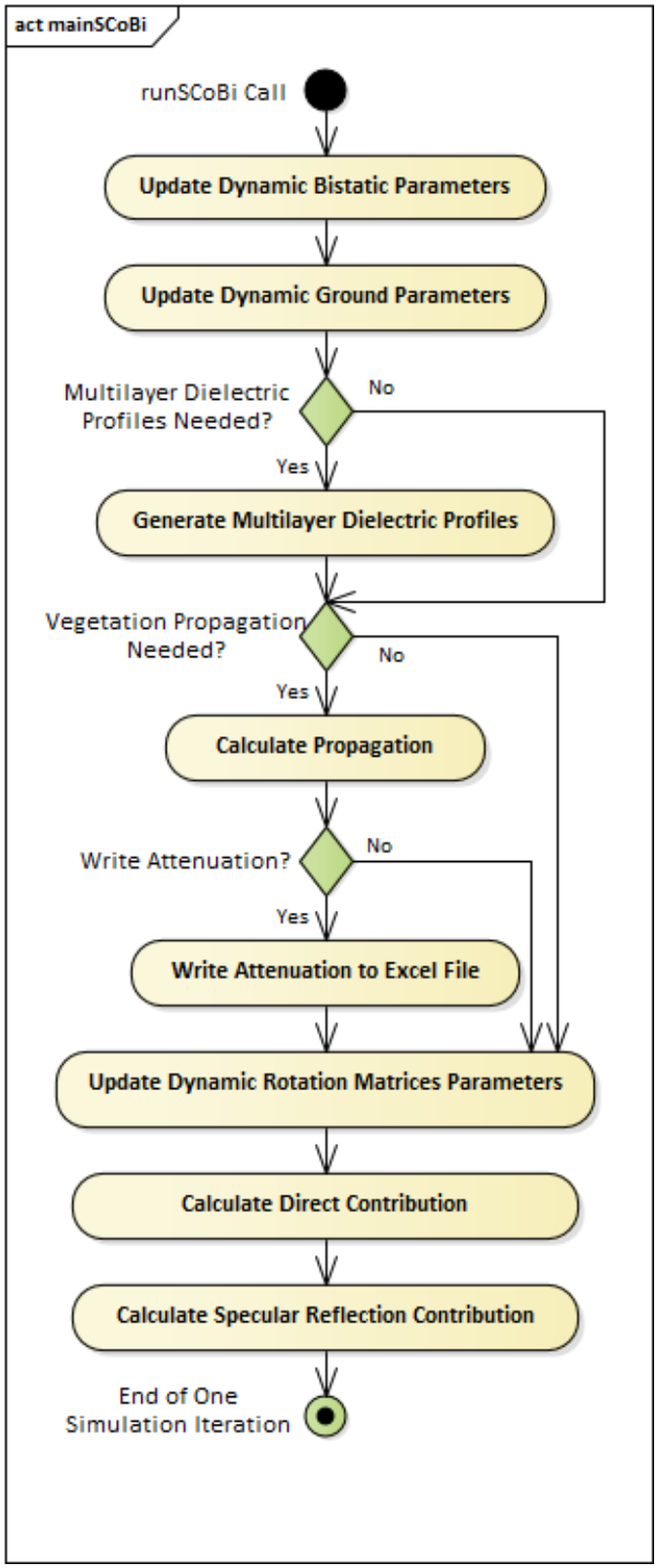

Fig. 5. mainSCoBi.m (Simulation iteration function) Activitiy Diagram

3. SCoBi Source Code Entities

The SCoBi source code entities (MATLAB functions and classes) are described here with respect to their packages under the ***lib*** package. The packages and their functions and classes here are listed alphabetically except the first three packages that are directly located under the ***lib*** package**.**

3.1. scobi Package

This package mostly consists of the sub-package that perform specific operations within the SCoBi simulator. The only code entitity that this package includes is the ***runSCoBi.m*** function.

i. Function ***runSCoBi***: It is the simulation engine function. *runSCoBi.m,*

- Starts the simulation,
- Calls GUI classes to get the user inputs,
- Performs input validity check by using the ***ParamsManager*** class's functions
- Runs the simulation iterations for the required number of simulations.

3.1.1. *bistatic* Package

This package consists of the functions that perform operations related to the bistatic geometry. Below are the functions of this package:

i. Function ***bistaticGeometry*****:** Calculates and outputs the bistatic geometry-related transformation matrices, direction vectors, and rotation matrices.

- It calls *calcFresnelZones* function to calculate the Fresnel zones for the specular reflection point and to use them in the calculation of the other points.

ii. Function ***calcFresnelZones*****:** Calculates and outputs the Fresnel zones for the specular reflection point and to use them in the calculation of the other points.

iii. Function ***transmitterGeometryManual*****:** Calculates the transmitter geometry-related transformation matrices.

iv. Function ***updateBistaticDynParams*****:** Calls the functions that calculate the bistatic geometry and transmitter parameters and updates the bistatic dynamic parameters (BistaticDynParams) with those parameter values in each simulation iteration.

- Calls ***transmitterGeometryManual*** function to calculate the transmitter geometry-related transformation matrices.
- Calls ***bistaticGeometry*** function to calculate the bistatic geometry-related transformation matrices, direction vectors, and rotation matrices.
- Updates ***BistaticDynParams*** class

v. Function ***updateRotMatDynParams*****:** Calculates the antenna polarization rotation matrices and updates ***RotMatDynParams*** with those values in each simulation iteration.

3.1.2. constants Package

This package contains the final static Constant and ConstantNames classes that contain global

variable that are used through the entire SCoBi simulator, and the Directories class that is used for tracking the source code directories.

i. Class ***Constants***: It contains the global constant variables. This class only consists of constant (final) properties that keep the global values for any purpose.

ii. Class ***ConstantNames*:** Global constant naming strings (char arrays). This class only consists of constant (final) properties that keep the global names for any purpose (e.g. files, variables, xml tags, etc.) in the simulation.

iii. Class ***Directories*:** This class allows the simulator to access to the source code and input directories. It has attributes for specific source code and input directories that are required to be accessed throughout the simulation. Every attribute can be reached by a static getter method.

3.1.3. ground Package

This package contains functions that perform ground-related calculations..

i. Function ***dielDobson***: It implements the Dobson ground dielectric model.

ii. Function ***dielMironov***: It implements the Mironov ground dielectric model.

iii. Function ***dielWang***: It implements the Wang ground dielectric model.

iv. Function ***reflectionCoeffSingle*:** Calculates the equivalent reflection coeff. of the rough ground from the Fresnel reflection coeff. of an avg. flat surface. Ground is considered as surface-only.

v. Function ***updateGndDynParams*:** Calculates the effective roughness parameter and calls the ground dielectric functions (Dobson, Mironow, Wang) to calculate the dielectric constant, and updates GndDynParams class with those values in each simulation iteration.

3.1.4. gui Package

i. Package ***images*:** It keeps the iamge files that are needed by the GUI windows.

ii. Package ***main***: It houses the GUI and handle classes for the *Analysis Selection Window*.

gui_SCoBiMain.fig is the GUI file that is created by using MATLAB's GUIDE tool.

gui_SCoBiMain.m is the auto-generated class for gui_SCoBiMain.fig.

gui_SCoBiMain_Manager: It's the handle class for the above GUI files. It implements the actions for GUI callbacks.

iii. Package ***scobi***: It houses the GUI and handle classes for the *Simulation Inputs Window*.

gui_SCoBi.fig is the GUI file that is created by using MATLAB's GUIDE tool.

gui_SCoBi.m is the auto-generated class for gui_SCoBi.fig.

gui_SCoBi_Manager: It's the handle class for the above GUI files. It implements the actions for GUI callbacks.

3.1.5. init Package

This package contains the functions that initializes the static parameters of the SCoBi simulations

by using the given inputs.

i. Function ***initAllInputParams***:This function calls the other parameter-initializer functions in an order to initialize the system parameters.

ii. Function ***initConfigParams****:* Fetches the configuration parameters' values from the input structure and initializes the *ConfigParams* class. It implements preprocessing of data beyond initialization (such as generating the combination of *Snapshot* simulation mode).

iii. Function ***initGndParams****:* Fetches the ground parameters' values from the input structure.

- Initializes the *GndParams* class.

iv. Function ***initRxParams****:* Fetches the receiver parameters' values from the input structure and initializes the *RxParams* class. It implements interpretation of inputs beyond initialization (such as deciding which antenna pattern is used) and initializing that as well.

- Calls the appropriate function to create the receiver antenna voltage pattern.

v. Function ***initRxGGParams****:* Fetches the parameters' values for the receiver with Generalized-Gaussian antenna pattern.

- Initializes the *RxGGParams* class.

vi. Function ***initRxUserDefinedParams****:* Fetches the parameters' values for the receiver with a custom (User-defied) antenna pattern.

- Initializes the *RxUserDefinedParams* class.

vii. Function ***initSimSettings****:* Fetches the simulation setting parameters' values from the input structure.

- Initializes the *simSettings* class.

viii. Function ***initTxParams****:* Fetches the transmitter parameters' values from the input structure.

- Initializes the *TxParams* class.

ix. Function ***initVegParams****:* Fetches the vegetation parameters' values, if any, from the input structure.

- Initializes the *VegParams* class.

x. Function ***initWithInputs****:* It's quite different than the other functions in this package. This function initializes the simulator with the recently initialized parameters.

- Initializes the simulation output directories.
- Saves input parameters for future reference.
- Calls a number of *ParamsManager* functions to determined ssimulation-related parameter
- Check the parameters' validity.

3.1.6. main Package

This package only contains the mainSCoBi.m function.

i. Function ***maisnSCoBi***: It performs every simulation iteration that is called by the runSCoBi.m function. *In every single iteration, it*

- Calls the update functions for the dynamic parameters,
- Calls ***multilayer*** and/or ***vegetation*** related functions,
- Calculates the SoOp products incrementally throughout the simulations.

3.1.7. param Package

This package contains the classes of the static and dynamic parameters of the system.

i. Class ***ParamsManager***: It is a static class that provides several functions for parameter management. It manages the parameters (which are constructed once from the inputs before simulations start held by static or dynamic parameter classes) throughout the entire simulation. There are two sub-packages under this package: dynamic and static

ii. Package ***dynamic***: This package contains the classes that hold dynamic parameters that are calculated and updated in every simulation iteration. Below are the dynamic parameter classes:

- Class ***BistaticDynParams***: This class houses the properties that are used for hbistatic geometry parameters such as the range, coordinate transformation matrices, propagation vectors, antenna rotation matrices, and significant points.
- Class ***GndDynParams***: Since the SCoBi simulator is using variable parameters such as volumetric soil moisture and roughness, ground parameters such as dielectric constant and effective surface roughness parameters change in every iteration. This changes require such parameters to be updated in every iteration. This Class holds those dynamic parameters.
- Class ***RotMatDynParams***: Similar to the GndDynParams class, antenna polarization rotation matrices should be calculated in each iteration because the antenna orientation might change thruugh the entire simulation.

iii. Package ***static***:This package consists of classes that are initialized at the beginning of the whole simulation and used as is.

- Class ***ConfigParams***: This class holds the multi-valued parameters that can be changing from iteration to iteration such as DoYs, transmitter orientation angles, VSM, and surface roughness.
- Class ***GndParams***: It keeps the static ground parameters such as the soilmixture information.
- Class ***RxParams***: It keeps the receiver antenna-related parameters such as its polarization, orientation, etc.
- Class ***RxGGParams***: It houses the parameters(e.g. beamwidth, side-lobe level, and cross-polarization level) that are specific to the Generalized-Gaussian receiver antenna.
- Class ***RxUserDefinedParams***: It houses the parameters that are specific to the User-defined receiver antenna.

- Class ***SimSettings:*** Significant simulation settings such as the campaign name, ground cover, and simulation mode.
- Class ***TxParams***: It keeps the transmitter antenna-related parameters such as its polarization, orientation, etc.

3.1.8. plot Package

This package is not in the simulator's regular flow; however, it can be briefly described here with the plotting functions: ***plotReflevtivityVsEL*** *and* ***plotReflectivityVsVSM****. These functions allw the user to plot the outputs of the simulations.*

3.1.9. products Package

This package contains the functions that calculate and store the simulated SoOp deliverables.

i. Function ***directTerm***: It calculates the direct (line-of-sight) received field and power between the transmitter and the receiver.
   - Stores the calculated values into simulation output folders in an incremental fashion as the simulation iterations continue.

ii. Function ***specularTerm***: It calculates the specular reflection coefficient (coherent bistatic forward scattering through the specular reflection point) and reflectivity.
   - Stores the calculated values into simulation output folders in an incremental fashion as the simulation iterations continue.

3.1.10. util Package

This package contains several utility functions that perfrorm from calculation of Muller matrices to writing to the file. The individual function details will not be given here.

3.2. vegetation Package

This package and its entities are dedicated to the simulations with the vegetation cover.

3.2.1. param Package

This package is similar to that of the **scobi** *package.It only contains the static sub-package and one class in it.*

i. Class ***VegParams:*** This class is only initialized if the ground cover is vegetation. It holds the parameters that are specific to vegetation layers such as dimensions, particle dielectrics.

3.2.2. propagation Package

This package is dedicated to the vegetation propagation calculations..

i. function ***calcPropagation***: This function calculates propagation and attenuation due to the vegetation layer, if any. Uses the functions eCylinder and eDisk to make calculations for dieelctric cylinder (stalk, branch, or needle) and dielectric disk (leaf)

- Stores the calculations into simulation output folders as metadata

ii. function ***eCylinder***: It handles the dielectric cylinder calculations.

iii. function ***eDisk***: It handles the dielectric disk calculations.

iv. function ***writeAttenuation***: This function writes the vegetation propagation results to an Excel file if selected through the simulation settings.

3.3. multilayer Package

This package consists of three sub-packages contain functions that handles the multi-layered ground structure.

1.1.1. ground Package

Similar to the ***\scobi\ground*** this package is responsible the multi-layered ground calculations.

i. Function ***generateDielMLProfiles***: Generates the selected dielctricprofiles (2nd order, 3rd order, logistic fit, discrete-slab) if the ground structure is Multi-layered.

- Updates the dynamic Multi-layered dielectric parameters (DielMLDynParams) with those parameters' values in each simulation iteration.
- Uses the information from GndMLParams and GndDynParams

ii. Function ***reflectionCoeffsML:*** Calculates the equivalent reflections coefficients of the multi-layered surface for the chosen dielectric profiles.

1.1.2. multidiel Package

This package is responsible for handling the reflection response of multilayered structure.

1.1.3. param Package

It is similar to the ***param*** package of the **scobi** package.This package holds the static and dynamic parameter classes for the SCoBi multilayer.