Module Interface Specification for RSSC

Xingzhi Liu

November 21, 2020

1 Revision History

Date	Version	Notes
Nov. 19, 2020	1.0	Initial Release

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/XingzhiMac/CAS741-Proj/

Contents

1 Revision History **i**

2 Symbols, Abbreviations and Acronyms **ii**

3 Introduction **1**

4 Notation **1**

5 Module Decomposition **1**

6 MIS of Control Module **3**
- 6.1 Module . . . 3
- 6.2 Uses . . . 3
- 6.3 Syntax . . . 3
  - 6.3.1 Exported Constants . . . 3
  - 6.3.2 Exported Access Programs . . . 3
- 6.4 Semantics . . . 3
  - 6.4.1 State Variables . . . 3
  - 6.4.2 Access Routine Semantics . . . 3

7 MIS of Input Parameters Module **5**
- 7.1 Module . . . 5
- 7.2 Uses . . . 5
- 7.3 Syntax . . . 5
  - 7.3.1 Exported Constants . . . 5
  - 7.3.2 Exported Access Programs . . . 5
- 7.4 Semantics . . . 5
  - 7.4.1 State Variables . . . 5
  - 7.4.2 Environment Variables . . . 6
  - 7.4.3 Assumptions . . . 6
  - 7.4.4 Access Routine Semantics . . . 6
  - 7.4.5 Local Functions . . . 8

8 MIS of Point Module **9**
- 8.1 Module . . . 9
- 8.2 Uses . . . 9
- 8.3 Syntax . . . 9
  - 8.3.1 Exported Constants . . . 9
  - 8.3.2 Exported Access Programs . . . 9
- 8.4 Semantics . . . 9
  - 8.4.1 State Variables . . . 9
  - 8.4.2 Environment Variables . . . 9

8.4.3 Assumptions . . . 9
8.4.4 Access Routine Semantics . . . 9
8.4.5 Local Functions . . . 10

9 MIS of Wall Module **11**
9.1 Module . . . 11
9.2 Uses . . . 11
9.3 Syntax . . . 11
9.3.1 Exported Constants . . . 11
9.3.2 Exported Access Programs . . . 11
9.4 Semantics . . . 11
9.4.1 State Variables . . . 11
9.4.2 Environment Variables . . . 12
9.4.3 Assumptions . . . 12
9.4.4 Access Routine Semantics . . . 12
9.4.5 Local Functions . . . 14

10 MIS of Floor Map Module **15**
10.1 Module . . . 15
10.2 Uses . . . 15
10.3 Syntax . . . 15
10.3.1 Exported Constants . . . 15
10.3.2 Exported Access Programs . . . 15
10.4 Semantics . . . 15
10.4.1 State Variables . . . 15
10.4.2 Environment Variables . . . 15
10.4.3 Assumptions . . . 15
10.4.4 Access Routine Semantics . . . 15
10.4.5 Local Functions . . . 16

11 MIS of Equation Finder **17**
11.1 Module . . . 17
11.2 Uses . . . 17
11.3 Syntax . . . 17
11.3.1 Exported Constants . . . 17
11.3.2 Exported Access Programs . . . 17
11.4 Semantics . . . 17
11.4.1 State Variables . . . 17
11.4.2 Environment Variables . . . 17
11.4.3 Assumptions . . . 17
11.4.4 Access Routine Semantics . . . 17
11.4.5 Local Functions . . . 18

12 MIS of Linear Signal Path Module **19**
12.1 Module 19
12.2 Uses 19
12.3 Syntax 19
12.3.1 Exported Constants 19
12.3.2 Exported Access Programs 19
12.4 Semantics 19
12.4.1 State Variables 19
12.4.2 Environment Variables 19
12.4.3 Assumptions 19
12.4.4 Access Routine Semantics 20
12.4.5 Local Functions 20

13 MIS of Intersection Module **21**
13.1 Module 21
13.2 Uses 21
13.3 Syntax 21
13.3.1 Exported Constants 21
13.3.2 Exported Access Programs 21
13.4 Semantics 21
13.4.1 State Variables 21
13.4.2 Environment Variables 21
13.4.3 Assumptions 21
13.4.4 Access Routine Semantics 21
13.4.5 Local Functions 22

14 MIS of Linear Path Loss Module **23**
14.1 Module 23
14.2 Uses 23
14.3 Syntax 23
14.3.1 Exported Constants 23
14.3.2 Exported Access Programs 23
14.4 Semantics 23
14.4.1 State Variables 23
14.4.2 Environment Variables 23
14.4.3 Assumptions 23
14.4.4 Access Routine Semantics 23
14.4.5 Local Functions 24

15 MIS of Line-Of-Sight Signal Module **25**
15.1 Module 25
15.2 Uses 25
15.3 Syntax 25

15.3.1 Exported Constants 25
15.3.2 Exported Access Programs 25
15.4 Semantics 25
15.4.1 State Variables 25
15.4.2 Environment Variables 25
15.4.3 Assumptions 25
15.4.4 Access Routine Semantics 25
15.4.5 Local Functions 26

16 MIS of Specular Reflection Module 27
16.1 Module 27
16.2 Uses 27
16.3 Syntax 27
16.3.1 Exported Constants 27
16.3.2 Exported Access Programs 27
16.4 Semantics 27
16.4.1 State Variables 27
16.4.2 Environment Variables 27
16.4.3 Assumptions 27
16.4.4 Access Routine Semantics 27
16.4.5 Local Functions 28

17 MIS of First-Order Reflection Signal Module 29
17.1 Module 29
17.2 Uses 29
17.3 Syntax 29
17.3.1 Exported Constants 29
17.3.2 Exported Access Programs 29
17.4 Semantics 29
17.4.1 State Variables 29
17.4.2 Environment Variables 29
17.4.3 Assumptions 29
17.4.4 Access Routine Semantics 30
17.4.5 Local Functions 30

18 MIS of Received Signal Strength Module 31
18.1 Module 31
18.2 Uses 31
18.3 Syntax 31
18.3.1 Exported Constants 31
18.3.2 Exported Access Programs 31
18.4 Semantics 31
18.4.1 State Variables 31

18.4.2 Environment Variables 31
18.4.3 Assumptions 31
18.4.4 Access Routine Semantics 31
18.4.5 Local Functions 32

19 MIS of Output Module 33
19.1 Module 33
19.2 Uses 33
19.3 Syntax 33
19.3.1 Exported Constants 33
19.3.2 Exported Access Programs 33
19.4 Semantics 33
19.4.1 State Variables 33
19.4.2 Environment Variables 33
19.4.3 Access Routine Semantics 33
19.4.4 Local Functions 33

20 MIS of Specification Parameters 34
20.1 Module 34
20.2 Uses 34
20.3 Syntax 34
20.3.1 Exported Constants 34
20.4 Semantics 34

3 Introduction

The following document details the Module Interface Specifications for Radio Signal Strength Calculator. It is intended to ease navigation through the program for design and maintenance purposes.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/XingzhiMac/CAS741-Proj/.

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 RSSC.

Data Type	**Notation**	**Description**
Boolean	Boolean	a 1-bit data with two possible values (0 and 1)
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, ∞)
non-negative integer	$\mathbb{N}_0$	a number without a fractional component in [0, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of RSSC uses some derived data types: sets, strings, and tuples. Sets are lists filled with elements of the same data type. In this document, a set of data in type T is represented as set[T]. Strings are lists of characters. Tuples contain a list of values, potentially of different types.

In addition, RSSC defines the following classes as its unique data types: Point (defined in section 8), Wall (defined in section 9), FloorMap (defined in section 10), LinearPath (defined in section 12), LineOfSight (defined in section 15), and FirstOrderReflection (defined in section 17).

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 Control Module Received Signal Strength Specification Parameters Module
Software Decision	Point Wall Floor Map Equation Finder Linear Signal Path Intersection Linear Path Loss Line-Of-Sight Signal Specular Reflection First Order Reflection Signal

Table 1: Module Hierarchy

6 MIS of Control Module

6.1 Module

main

6.2 Uses

Param (section 7), FloorMap(section 10), ReceivedSignalStrength(section 18), Output(section 19)

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 Access Routine Semantics

main():

- transition: Modify the state of Param module and the environment variables for the Output module by following these steps:

  Get (filenameTSM: string), (filenameSP: string), (filenameWALL: string), and (filenameOut: string) from user

  Param.load_params(filenameTSM, filenameSP, filenameWALL)

  map = FloorMap.create()

  $[P_{sp}^{dBm}]$:= empty set
  For ($sampling\_point$: Point) in Param.$[Pos_{sp}]$:
  P_{sp}^{dBm} := ReceivedSignalStrength.get_received_power(map, $sampling\_point$)
  Add P_{sp}^{dBm} into the set $[P_{sp}^{dBm}]$

Output.output(filenameOut, Param.$[Pos_{sp}]$, $[P_{sp}^{dBm}]$)

7 MIS of Input Parameters Module

7.1 Module

Param

7.2 Uses

SpecParam (section 20), Point (section 8)

7.3 Syntax

7.3.1 Exported Constants

None

You never use this outside of this module. Consider making this a local function.

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
load_params	s1: string, s2: string, s3: string	-	FileError
verify_params	-	-	badTransmittance, badReflectance, badTransmissionPower, badSignalFrequency, badPosition, inconsistentWallParams
Pos_{tsm}	-	Point	-
$[Pos_{sp}]$	-	set[Point]	-
$[C]$	-	set[Point]	-
$[D]$	-	set[Point]	-
$[T]$	-	set[$\mathbb{R}$]	-
$[R]$	-	set[$\mathbb{R}$]	-
P_{tsm}^{dBm}	-	$\mathbb{R}$	-
f	-	$\mathbb{R}$	-

7.4 Semantics

7.4.1 State Variables

Pos_{tsm}: Point
$[Pos_{sp}]$: set[Point]
$[C]$: set[Point]
$[D]$: set[Point]

$[T]$: set[$\mathbb{R}$]
$[R]$: set[$\mathbb{R}$]
P_{tsm}^{dBm}: $\mathbb{R}$
f: $\mathbb{R}$
$length_C$: $\mathbb{R}$
$length_D$: $\mathbb{R}$
$length_T$: $\mathbb{R}$
$length_R$: $\mathbb{R}$

7.4.2 Environment Variables

tsmFile: set[string]
spFile: set[string]
wallFile: set[string]

7.4.3 Assumptions

- load_params will be called before any of the state variables be accessed.
- tsmFile contains the string equivalents of the numeric values for Pos_{tsm}, P_{tsm}^{dBm} and f, each on a new line.
- spFile contains the string equivalents of elements in the user-input item $[Pos_{sp}]$, each on a new line, in the form of two numbers separated with a comma.
- wallFile contains the string equivalents of elements in the user-input items $[C]$, $[D]$, $[T]$, and $[R]$. Each line is in the form of 6 numbers separated by 5 commas (each line should be "$x_{C_x}, y_{C_x}, x_{D_x}, y_{D_x}, T_x, R_x$").

7.4.4 Access Routine Semantics

Param.Pos_{tsm}():

- output: $out := Pos_{tsm}$
- exception: none

Param.$[Pos_{sp}]$():

- output: $out := [Pos_{sp}]$
- exception: none

Param.$[C]$():

- output: $out := [C]$
- exception: none

Param.$[D]()$:

- output: $out := [D]$
- exception: none

Param.$[T]()$:

- output: $out := [T]$
- exception: none

Param.$[R]()$:

- output: $out := [R]$
- exception: none

Param.$P_{tsm}^{dBm}()$:

- output: $out := P_{tsm}^{dBm}$
- exception: none

Param.$f()$:

- output: $out := f$
- exception: none

Param.$length_C()$:

- output: $out :=$ number of elements in the user-input item $[C]$
- exception: none

Param.$length_D()$:

- output: $out :=$ number of elements in the user-input item $[D]$
- exception: none

Param.$length_T()$:

- output: $out :=$ number of elements in the user-input item $[T]$
- exception: none

Param.$length_R$():

- output: $out :=$ number of elements in the user-input item $[R]$
- exception: none

load_params(s1: string, s2: string, s3: string):

- transition:
  The file names s1, s2, and s3 are associated with tsmFile, spFile, and wallsFile respectively.
  The state variables are modified with the following procedures:
  1. Read data from the three files to populate the state variables from **??** (from Pos_{tsm} to f).
  2. Store the lengths of $[C], [D], [T], and [R]$ as $length_C, length_D, length_T$, and $length_R$ respectively.
  3. verify_params()

  ?

- exception: $exc :=$ any of the file names (s1, s2, or s3) cannot be found OR of any file's format (tsmFile, spFile, or wallsFile) is incorrect $\Rightarrow$ FileError

verify_params():

- output: $out :=$ none
- exception: $exc :=$
  $\neg(T_{min} \leq T_x \leq T_{max} \ \forall \ T_x \in [T]) \Rightarrow$ badTransmittance
  $\neg(R_{min} \leq R_x \leq R_{max} \ \forall \ R_x \in [R]) \Rightarrow$ badReflectance
  $\neg(P_{min}^{dBm} \leq P_{tsm}^{dBm} \leq P_{max}^{dBm}) \Rightarrow$ badTransmissionPower
  $\neg(f_{min} \leq f \leq f_{max}) \Rightarrow$ badSignalFrequency
  $\neg(x_{min} \leq x_{C_x} \leq x_{max} \ \forall \ C_x \in [C]) \Rightarrow$ badPosition
  $\neg(y_{min} \leq y_{C_x} \leq y_{max} \ \forall \ C_x \in [C]) \Rightarrow$ badPosition
  $\neg(x_{min} \leq x_{D_x} \leq x_{max} \ \forall \ D_x \in [D]) \Rightarrow$ badPosition
  $\neg(y_{min} \leq y_{D_x} \leq y_{max} \ \forall \ D_x \in [D]) \Rightarrow$ badPosition
  $\neg(length_C = length_D = length_T = length_R) \Rightarrow$ inconsistentWallParams

7.4.5 Local Functions

None

8 MIS of Point Module

8.1 Module

Point

8.2 Uses

None

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
create	x_coordinate: $\mathbb{R}$, y_coordinate: $\mathbb{R}$	Point	-
set_coordinates	x_coordinate: $\mathbb{R}$, y_coordinate: $\mathbb{R}$	-	-
get_coordinates	-	$x : \mathbb{R}, y : \mathbb{R}$	-

8.4 Semantics

8.4.1 State Variables

$x : \mathbb{R}$
$y : \mathbb{R}$

8.4.2 Environment Variables

None

8.4.3 Assumptions

None

8.4.4 Access Routine Semantics

create(x_coordinate: $\mathbb{R}$, y_coordinate: $\mathbb{R}$):

- transition:
  x := x_coordinate
  y := y_coordinate
- output: out := self

set_coordinates(x_coordinate: $\mathbb{R}$, y_coordinate: $\mathbb{R}$):

- transition:
  x := x_coordinate
  y := y_coordinate
- output: none

get_coordinates():

- output: $out := x, y$

8.4.5 Local Functions

None

9 MIS of Wall Module

Template?

It seems to me that you are using wall as an ADT

9.1 Module

Wall

9.2 Uses

Point (section 8), EquationFinder (section 11)

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
create	start: Point, end: Point	Wall	invalidWall
set_start_point	Point	-	invalidWall
set_end_point	Point	-	invalidWall
get_start_point	-	Point	-
get_end_point	-	Point	-
set_transmittance	$\mathbb{R}$	-	-
set_reflectance	$\mathbb{R}$	-	-
get_transmittance	-	$\mathbb{R}$	-
get_reflectance	-	$\mathbb{R}$	-
get_unit_normal	-	$n1 : \mathbb{R}$, $n2 : \mathbb{R}$	-
get_line_equation	-	$m1 : \mathbb{R}$, $m2 : \mathbb{R}$, $k : \mathbb{R}$	-

Rather than get_transmittance, a better name for the getter might be Tx?

9.4 Semantics

9.4.1 State Variables

C_x : Point
D_x : Point
$T_x : \mathbb{R}$
$R_x : \mathbb{R}$
$m1 : \mathbb{R}$
$m2 : \mathbb{R}$
$k : \mathbb{R}$

$n1 : \mathbb{R}$
$n2 : \mathbb{R}$

9.4.2 Environment Variables

None

9.4.3 Assumptions

None

9.4.4 Access Routine Semantics

create(start: Point, end: Point, transmittance: $\mathbb{R}$, reflectance: $\mathbb{R}$):

- transition:
  $C_x :=$ start
  $D_x :=$ end
  $T_x :=$ transmittance
  $R_x :=$ reflectance

  Use Equation Finder Module to find equation parameters:
  $m1, m2, k :=$ EquationFinder.find_equation(start, end)

  Find the unit normal vector:
  $n1, n2 :=$ find_unit_normal(C_x, D_x) nice use of local functions

- output: $out :=$ self
- exception: $exc := C_x$ and D_x have the same coordinates $\Rightarrow$ invalidWall

this is fine, but even better if you say it mathematically

set_start_point(start: Point):

- transition:
  $C_x :=$ start

  Use Equation Finder Module to find equation parameters:
  $m1, m2, k :=$ EquationFinder.find_equation(C_x, D_x)

  Find the unit normal vector:
  $n1, n2 :=$ find_unit_normal(C_x, D_x)

- output: none
- exception: $exc := C_x$ and D_x have the same coordinates $\Rightarrow$ invalidWall

set_end_point(end: Point):

- transition:
  $D_x :=$ end

  Use Equation Finder Module to find equation parameters:
  $m1, m2, k :=$ EquationFinder.find_equation(C_x, D_x)

  Find the unit normal vector:
  $n1, n2 :=$ find_unit_normal(C_x, D_x)

- output: none
- exception: $exc := C_x$ and D_x have the same coordinates $\Rightarrow$ invalidWall

get_start_point():

- output: $out := C_x$
- exception: none

get_end_point():

- output: $out := D_x$
- exception: none

set_transmittance(transmittance: $\mathbb{R}$):

- transition: $T_x :=$ transmittance
- output: none
- exception: none

set_reflectance(resistance: $\mathbb{R}$):

- transition: $R_x :=$ resistance
- output: none
- exception: none

get_transmittance():

- output: $out := T$
- exception: none

get_reflectance():

- output: $out := R$
- exception: none

get_unit_normal():

- output: $out := n1, n2$
- exception: none

get_line_equation():

- output: $out := m1, m2, k$
- exception: none

9.4.5 Local Functions

find_unit_normal(C_x, D_x):

- transition:
  $x_{C_x}, y_{C_x} := C_x$.get_coordinates()
  $x_{D_x}, y_{D_x} := D_x$.get_coordinates()
  $\begin{bmatrix} n1 & n2 \end{bmatrix} := \begin{bmatrix} (x_{D_x} - x_{C_x}) & (y_{D_x} - y_{C_x}) \end{bmatrix} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \cdot \frac{1}{\sqrt{(y_{D_x} - y_{C_x})^2 + (x_{D_x} - x_{C_x})^2}}$
- output: out $:= n1, n2$
- exception: none

10 MIS of Floor Map Module

10.1 Module

FloorMap

10.2 Uses

Param(section 7), Point(section 8), Wall(section 9)

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
create	-	FloorMap	-
get_wall	$\mathbb{N}_0$	Wall	invalidIndex
get_map	-	FloorMap	-

10.4 Semantics

10.4.1 State Variables

wall_list: set[Wall]
wall_list_length: $\mathbb{N}_0$

the name wall_list suggests a sequence. Do you mean a set or a sequence? If you need access to the ith wall, you would have a sequence. If you don't need to pick distinct walls, a set is fine.

10.4.2 Environment Variables

None

10.4.3 Assumptions

create() will be called before the FloorMap object can be accessed.

10.4.4 Access Routine Semantics

create():

- transition:
  Get parameters of all walls from Input Parameters Module (section 7):
  Let $list_C$ = Param.$[C]()$;

Let $list_D$ = Param.$[D]()$;
Let $list_T$ = Param.$[T]()$;
Let $list_R$ = Param.$[R]()$;
Then:
wall_list_length := Param.$length_C()$
Then create wall_list as such:
for i = {0, 1, 2, ... , (wall_list_length - 1)},
wall_list(i) := Wall.create($list_C(i), list_D(i), list_T(i), list_R(i)$)

- output: out := self
- exception: none

get_wall(index: $\mathbb{N}_0$):

- output: out := wall_list(index)
- exception: exc := $\neg$(0 $\leq$ index $\leq$ wall_list_length - 1) $\Rightarrow$ invalidIndex

get_map():

- output: out := self
- exception: none

10.4.5 Local Functions

None

11 MIS of Equation Finder

11.1 Module

EquationFinder

11.2 Uses

Point(section 8)

11.3 Syntax

11.3.1 Exported Constants

None

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
find_equation	start: Point, end: Point	m1: $\mathbb{R}$, m2: $\mathbb{R}$, k: $\mathbb{R}$	-

11.4 Semantics

11.4.1 State Variables

None

11.4.2 Environment Variables

None

11.4.3 Assumptions

None

11.4.4 Access Routine Semantics

find_equation(start: Point, end: Point):

- output:
  Let x_{C_x}, y_{C_x} = start.get_coordinates();
  Let x_{D_x}, y_{D_x} = end.get_coordinates();
  Then
  $$m1 := \begin{cases} -\frac{y_{D_x} - y_{C_x}}{x_{D_x} - x_{C_x}} & \text{if } x_{D_x} - x_{C_x} \neq 0 \\ 1 & \text{else} \end{cases}$$

$$m2 := \begin{cases} 1 & \text{if } x_{D_x} - x_{C_x} \neq 0 \\ 0 & \text{else} \end{cases}$$
$$k := m1 \cdot x_{C_x} + m2 \cdot y_{C_x} = m1 \cdot x_{D_x} + m2 \cdot y_{D_x}$$
$$out := m1, m2, k$$

- exception: none

11.4.5 Local Functions

None

12 MIS of Linear Signal Path Module

12.1 Module

LinearPath

12.2 Uses

Point (section 8), EquationFinder (section 11)

12.3 Syntax

12.3.1 Exported Constants

None

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
create	start: Point, end: Point	LinearPath	-
get_equation	-	$m1 : \mathbb{R}$, $m2 : \mathbb{R}$, $k : \mathbb{R}$	-
get_start_point	-	Point	-
get_end_point	-	Point	-
get_length	-	$\mathbb{R}$	-

12.4 Semantics

12.4.1 State Variables

E: Point
F: Point
$m1 : \mathbb{R}$
$m2 : \mathbb{R}$
$k : \mathbb{R}$
$length : \mathbb{R}$

12.4.2 Environment Variables

None

12.4.3 Assumptions

None

12.4.4 Access Routine Semantics

create(start: Point, end: Point):

- transition:
  E := start
  F := end

  Use Equation Finder Module to find the path's equation parameters:
  $m1, m2, k$:= EquationFinder.find_equation(E, F)

  Let x_E, y_E = E.get_coordinates();
  Let x_F, y_F = F.get_coordinates();
  The find the physical length of the linear path:
  $length := \sqrt{(x_E - x_F)^2 + (y_E - y_F)^2}$

- output: out := self

- exception: none

get_equation():

- output: $out := m1, m2, k$

- exception: none

get_start_point():

- output: $out := E$

- exception: none

get_end_point():

- output: $out := F$

- exception: none

get_length():

- output: $out := length$

- exception: none

12.4.5 Local Functions

None

13 MIS of Intersection Module

13.1 Module

Intersection

13.2 Uses

Point (section 8), Wall (section 9), LinearPath (section 12)

13.3 Syntax

13.3.1 Exported Constants

None

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
find_intersection	Wall, LinearPath	Point	-
is_valid	Wall, LinearPath	Boolean	-

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

find_intersection($wall$: Wall, $path$: LinearPath):

- output:
  Let $M = \begin{bmatrix} wall.m1 & wall.m2 \\ path.m1 & path.m2 \end{bmatrix}$;

Let $K = \begin{bmatrix} wall.k \\ path.k \end{bmatrix}$;

$t' :=$ Point.create(x, y) such that $M \begin{bmatrix} x \\ y \end{bmatrix} = K$, if $det(M) \neq 0$; or

$t' :=$ Point.create(0, 0), if $det(M) = 0$.

$out := t'$

- exception: none

is_valid($wall$: Wall, $path$: LinearPath):

- output:
  Let $M = \begin{bmatrix} wall.m1 & wall.m2 \\ path.m1 & path.m2 \end{bmatrix}$;
  Let $K = \begin{bmatrix} wall.k \\ path.k \end{bmatrix}$;

  If $det(M) \neq 0$,
  $t' :=$ Point.create(x, y) such that $M \begin{bmatrix} x \\ y \end{bmatrix} = K$
  If
  $\max(\min(wall.C_x.x, wall.D_x.x), \min(path.C_x.x, path.D_x.x)) < t'.x < min(max(wall.C_x.x, wall.D_x.x$
  and
  $\max(\min(wall.C_x.y, wall.D_x.y), \min(path.C_x.y, path.D_x.y)) < t'.x < min(max(wall.C_x.y, wall.D_x.y)$
  $Ind := 1$;
  otherwise $Ind := 0$.
  If $det(M) = 0$,
  $Ind := 0$.

  $out := Ind$

- exception: none

You could use a conditional expression for this
$(C_1 \Rightarrow r_1 \mid C_2 \Rightarrow r_2 \mid \ldots \mid C_n \Rightarrow r_n)$
(described in Hoffman & Strooper

13.4.5 Local Functions

None

14 MIS of Linear Path Loss Module

14.1 Module

LinearLoss

14.2 Uses

FloorMap (section 10), LinearPath (section 12), Intersection (section 13)

14.3 Syntax

14.3.1 Exported Constants

None

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
find_linear_path_loss	path:LinearPath, f: $\mathbb{R}$	$\mathbb{R}$	-

14.4 Semantics

14.4.1 State Variables

None

14.4.2 Environment Variables

None

14.4.3 Assumptions

FloorMap.create() will be called before calling FloorMap.get_map().

14.4.4 Access Routine Semantics

find_linear_path_loss ($path$: LinearPath, $freq : \mathbb{R}$):

- output:
  The output is dependent on two parameters: $FSPL$ and T_{total}.
  Update $FSPL$:
  map: FloorMap.get_map()
  $FSPL := (\frac{4\pi \cdot path.length}{3 \times 10^8})^2$

Is this the length of the wall? the wall thickness? area?

This implies wall is a sequence in the notation. If you are using wall as a set, the notation would be something like

$*(w : wall \mid w \in walllist : w.\text{getrelevant}^{?} * w.T_x)$

Update T_{total}:
$T_{total} := \Pi_{x=0}^{N_w}(wall_x \cdot T_x^{Ind_{t,x}})$
Where
N_w := *map*.wall_list_length - 1
$wall_x$:= map.get_wall(x)
$Ind_{t,x}$:= Intersection.is_valid($wall_x, path$)

$out := \frac{T_{total}}{FSPL}$

- exception: none

14.4.5 Local Functions

None

wall $* T_x$ looks like a type mismatch

You seem to mix whether wall is a set or a sequence. Make a choice and make it consistent throughout

15 MIS of Line-Of-Sight Signal Module

15.1 Module

LineOfSight

15.2 Uses

Param (section 7), Point (section 8), LinearPath (section 12), LinearLoss (section 14)

15.3 Syntax

15.3.1 Exported Constants

$\phi_{LOS} := 0$

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
create	sampling_point: Point	LineOfSight	-
get_path_length	-	$\mathbb{R}$	-
get_amplitude	-	$\mathbb{R}$	-
get_phase_angle	-	$\mathbb{R}$	-

15.4 Semantics

15.4.1 State Variables

Pos_{sp}: Point
$d_{tsm,sp}$: $\mathbb{R}$
P_{LOS}: $\mathbb{R}$

15.4.2 Environment Variables

None

15.4.3 Assumptions

None

15.4.4 Access Routine Semantics

create(sampling_point: Point):

- transition:
  Update Pos_{sp}:
  Pos_{sp} := sampling_point

  Update $d_{tsm,sp}$:
  $freq$:= Param.f()
  Pos_{tsm} := Param.Pos_{tsm}()
  $path$:= LinearPath.create(Pos_{tsm}, Pos_{sp})
  $d_{tsm,sp}$:= $path$.get_length()

  Update P_{LOS}:
  P_{tsm}^{dBm} := Param.P_{tsm}^{dBm}()
  $P_{tsm} := 10^{\frac{P_{tsm}^{dBm}-30}{10}}$
  $P_{LOS} := P_{tsm}\cdot$ LinearLoss.find_linear_path_loss($path, freq$)

- output: out := self

- exception: none

get_path_length():

- output: $out := d_{tsm,sp}$

- exception: none

get_amplitude():

- output: $out := P_{LOS}$

- exception: none

get_phase_angle():

- output: $out := \phi_{LOS}$

- exception: none

15.4.5 Local Functions

None

16 MIS of Specular Reflection Module

16.1 Module

Specular

16.2 Uses

Point (section 8), Wall (section 9), LinearPath (section 12), Intersection (section 13)

16.3 Syntax

16.3.1 Exported Constants

None

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
get_mirrored_paths	$wall_x$:Wall, $start$:Point, end:Point	$path_{RS1}$:LinearPath, $path_{RS2}$:LinearPath, $Ind_{r,x}$: Boolean	-

16.4 Semantics

16.4.1 State Variables

None

16.4.2 Environment Variables

None

16.4.3 Assumptions

None

16.4.4 Access Routine Semantics

get_mirrored_paths($wall_x$:Wall, $start$:Point, end:Point):

- output:
  Let $n := \begin{bmatrix} wall_x.n1 & wall_x.n2 \end{bmatrix}$;
  Let $t := \begin{bmatrix} wall_x.C_x.x & wall_x.C_x.y \end{bmatrix}$;
  Let $p := \begin{bmatrix} start.x & start.y) \end{bmatrix}$;
  Then solve

$\begin{bmatrix} p'_x & p'_y \end{bmatrix} = p - 2n(n \cdot (p - t))$
for the values of p'_x and p'_y.
let p' := Point.create(p'_x, p'_y);
Let $mirrored\_path$:= LinearPath.create(p', end);
Then
t' := Intersection.find_intersection($wall_x$, $mirrored\_path$);
$Ind_{r,x}$:= Intersection.is_valid($wall_x$, $mirrored\_path$);
$path_{RS1}$:= LinearPath.create(p, t');
$path_{RS2}$:= LinearPath.create(t', end);
out := $path_{RS1}$, $path_{RS2}$, $Ind_{r,x}$

- exception: none

16.4.5 Local Functions

None

17 MIS of First-Order Reflection Signal Module

17.1 Module

FirstOrderReflection

17.2 Uses

Param (section 7), Wall (section 9), LinearLoss (section 12), LineOfSight (section 15), Specular (section 16)

17.3 Syntax

17.3.1 Exported Constants

None

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
create	$wall_x$: Wall, $sampling\_point$: Point, LOS: LineOfSight	FirstOrderReflection	-
get_amplitude	-	$\mathbb{R}$	-
get_phase_angle	-	$\mathbb{R}$	-

17.4 Semantics

17.4.1 State Variables

Pos_{sp}: Point
$path_{RS1}$: LinearPath
$path_{RS2}$: LinearPath
$Ind_{r,x}$: Boolean
P_{FORS}: $\mathbb{R}$
ϕ_{FORS}: $\mathbb{R}$

17.4.2 Environment Variables

None

17.4.3 Assumptions

None

17.4.4 Access Routine Semantics

create($wall_x$:Wall, sampling_point: Point, LOS: LineOfSight):

- transition:
  Update Pos_{sp}:
  $Pos_{sp} := sampling\_point$

  Update $path_{RS1}$, $path_{RS2}$, and $Ind_{r,x}$:
  $freq :=$ Param.$f()$
  $Pos_{tsm} :=$ Param.$Pos_{tsm}()$
  $path_{RS1}$, $path_{RS2}$, $Ind_{r,x} :=$ Specular.get_mirrored_paths$(wall_x, Pos_{tsm}, Pos_{sp})$

  Update P_{FORS}:
  If $Ind_{r,x} = 1$:
  $P_{tsm}^{dBm} :=$ Param.$P_{tsm}^{dBm}()$
  $P_{tsm} := 10^{\frac{P_{tsm}^{dBm}-30}{10}}$
  $transmittance_{RS1} :=$ LinearLoss.find_linear_path_loss$(path_{RS1}, freq)$
  $transmittance_{RS2} :=$ LinearLoss.find_linear_path_loss$(path_{RS2}, freq)$
  $P_{FORS} := P_{tsm} \cdot transmittance_{RS1} \cdot transmittance_{RS2} \cdot wall_x$.get_reflectance()
  Otherwise:
  $P_{FORS} := 0$

  Update ϕ_{FORS}:
  If $Ind_{r,x} = 1$:
  $\phi_{FORS} := 2\pi f \frac{path_{RS1}.length + path_{RS2}.length - LOS.d_{tsm,sp}}{3 \times 10^8}$ Otherwise:
  $\phi_{FORS} := 0$

- output: $out :=$ self

- exception: none

get_amplitude():

- output: $out := P_{FORS}$

- exception: none

get_phase_angle():

- output: $out := \phi_{FORS}$

- exception: none

17.4.5 Local Functions

None

18 MIS of Received Signal Strength Module

18.1 Module

ReceivedSignalStrength

18.2 Uses

FloorMap (section 10), LineOfSight (section 15), FirstOrderReflection (section 17)

18.3 Syntax

18.3.1 Exported Constants

None

18.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
get_received_power	*map*: FloorMap, *sampling_point*: Point	$\mathbb{R}$	invalidReceivedStrength

18.4 Semantics

18.4.1 State Variables

None

18.4.2 Environment Variables

None

18.4.3 Assumptions

FloorMap.create() will be called before calling FloorMap.get_map() in this module.

18.4.4 Access Routine Semantics

get_received_power(*map*: FloorMap, *sampling_point*: Point):

- output:
  Get Floor Map:
  map := FloorMap.get_map()
  map_complexity := *map*.wall_list_length - 1

Find Line-Of-Sight Signal
$LOS :=$ LineOfSight.create($sampling\_point$)

Find First-Order reflection signals by wall:
For x in (0, 1, 2, ... , $map\_complexity$):
$wall_x := map$.get_wall(x)
$FORS(x) :=$ FirstOrderReflection.create($wall_x, sampling\_point, LOS$)

Find total received signal:
$P_{LOS} := LOS$.get_amplitude()
$\phi_{LOS} := LOS$.get_phase_angle()
$P_{FORS_x} := FORS(x)$.get_amplitude()
$\phi_{FORS_x} := FORS(x)$.get_phase_angle()
$P_{sp}\angle\phi_{sp} := P_{LOS}\angle\phi_{LOS} + \sum_{x=0}^{map\_complexity} P_{FORS_x}\angle\phi_{FORS_x}$
$P_{sp}^{dBm} := 30 + 10\log_{10}(P_{sp})$
$out := P_{sp}^{dBm}$

- exception: $exc := (P_{tsm}^{dBm} \leq P_{sp}^{dBm}) \Rightarrow$ invalidReceivedStrength

18.4.5 Local Functions

None

19 MIS of Output Module

19.1 Module

Output

19.2 Uses

Param (section 7), Point (section 8)

19.3 Syntax

19.3.1 Exported Constants

None

19.3.2 Exported Access Programs

Name	In	Out	Exceptions
output	fname: string, $[Pos_{sp}]$: set[Point], $[P_{sp}^{dBm}]$: set[$\mathbb{R}$]	-	-

19.4 Semantics

19.4.1 State Variables

None

19.4.2 Environment Variables

file: a text file

19.4.3 Access Routine Semantics

output(fname, $[Pos_{sp}]$, $[P_{sp}^{dBm}]$):

- transition: write to environment variable named fname the calculated received signal strengths $[P_{sp}^{dBm}]$ and their corresponding sampling points in $[Pos_{sp}]$.
  Each line of the output file will be 3 numbers separated by comma: $[Pos_{sp}.x, Pos_{sp}.y, P_{sp}^{dBm}]$.
- exception: none

19.4.4 Local Functions

None

20 MIS of Specification Parameters

20.1 Module

SpecParam

20.2 Uses

None

20.3 Syntax

20.3.1 Exported Constants

From ?? in SRS P_{max}^{dBm} := 15 (dBm)
P_{min}^{dBm} := -30 (dBm)
f_{min} := 30 (Hz)
f_{max} := 3×10^{11} (Hz)
x_{min} := -20 (m)
x_{max} := 20 (m)
y_{min} := -20 (m)
y_{max} := 20 (m)

20.4 Semantics

N/A

Great work!
I think you will find inconsistencies when you do more of the implementation and the formalization could be improved, but the design looks good.

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.