Module Interface Specification for Sun Catcher

Sharon (Yu-Shiuan) Wu

December 5, 2019

1 Revision History

Date	Version	Notes
2019/11/25	1.0	First Version
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/sharyuwu/optimum-tilt-of-solar-panels/blob/master/docs/SRS/SRS.pdf

[Also add any additional symbols, abbreviations or acronyms —SS]

Contents

3 Introduction

The following document details the Module Interface Specifications for Sun Catcher[Fill in your project name and description —SS]

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/sharyuwu/optimum-tilt-of-solar-panels. [provide the url for your repo —SS]

4 Notation

[You should describe your notation. You can use what is below as a starting point. —SS]

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 Sun Catcher.

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 (-∞, ∞)
bool	$\mathbb{B}$	a statement of True or False

The specification of Sun Catcher 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, Sun Catcher 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	Control Module Input Parameters Module Input Verification Module Output Parameters Module Solar Energy Absorption Module Optimum Tilt Angle Module Sun Intensity Equation Module Zenith Angle Equation Module
Software Decision Module	Day ADT Module Table-layout Module Sun Catcher Type Module

Table 1: Module Hierarchy

6 MIS of Control Module

6.1 Module

Control

6.2 Uses

InputPara9 [Please leave space between the name and the reference. You should also identify what type of reference it is, like you would for a Figure or a Table. That is, you should say (Section 9). —SS]
InputVer (Section 10)
OutputPara (Section 11)
TiltAng (Section 13)
Energy (Section 14)
SunInten (Section 15)
ZenithAng (Section 16)

6.3 Syntax

6.3.1 Exported Constants

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
mian [main? —SS]	-	-	-

6.4 Semantics

6.4.1 State Variables

[You use this type, but I don't think you import (use) the module that defines it —SS] [Change DayT as Day ADT —Author] [State Variable dayS and dayE should only exit in Input Parameters Module —Author]

[Why are these two state variables here? I thought you were going to use the Input Parameters module? —SS] [Now that I think about it, given that you are planning on implementing with Haskell, you might want to avoid having modules with state information. —SS]

6.4.2 Environment Variables

6.4.3 Assumptions

6.4.4 Access Routine Semantics

main ( ):

- transition: Implement the InputPara and the environment variables for Output by following steps.

  Get (Φ_P: DegreeT, P_{A_h}: real, P_{A_w}: real, $year_\mathrm{Start}$: natural number, $month_\mathrm{Start}$: natural number, day_Start: natural number, $year_\mathrm{End}$: natural number, $month_\mathrm{End}$: natural number, day_End: natural number) and (filename: string) from users' input.

  load_anale_data (filename)
  InputPara.init ( )

  # Verify the input values using Input Verfication Module
  verified (Φ_P, P_{A_h}, P_{A_w}, dayS, dayE)

  # Get the zenith angle between dayS and dayE
  θ_{S_date} = getzen ( )

  # Get the optimum tilt angle using θ_{S_date}
  setilt (θ_{S_date})
  θ_T = getilt ( )

  # Get the estimated solar absorption using Optimum Tilt Angle Module
  setenergy (P_{A_w}, P_{A_h}, getmaxInten ( ), θ_{S_date})
  P_E = getavenergy ( )

  # Output P_E and θ_T to the Output paramaters Module
  addresult (θ_T, P_E)

  # Display the output values on the screen using the Table-layout Module
  setable (getresult ( ))
  display ( )

[It is nice to have a newpage between modules. —SS]

7 Sun Catcher Type Module

7.1 Module

SunCatTy

7.2 Uses

N/A

7.3 Exported Types

DegreeT = $\mathbb{R}$
AnalemmaT = a tuple of (x: $\mathbb{R}$,y: $\mathbb{R}$, z: $\mathbb{R}$)

7.4 Syntax

7.4.1 Exported Constants

N/A

7.4.2 Exported Access Programs

N/A

7.5 Semantics

N/A

7.5.1 State Variables

N/A

7.5.2 Environment Variables

N/A

7.5.3 Assumptions

For DayT the day will always lie between 1 and 31, the month will always lie between 1 and 12, and year will always be greater than 1.

8 Day ADT Module

8.1 Template Module

DayT

8.2 Uses

SunCatTy7

8.3 Exported Types

DayT = ? [It seems confusing to me that you have a DayT, that is not an ADT, and a DayDurT that is an ADT. Couldn't you just have an ADT for DayT? —SS]

8.4 Syntax

8.4.1 Exported Constants

8.4.2 Exported Access Programs

Name	In	Out	Exceptions
new DayT	$\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$	DayT	invalid_argument
getnext	-	DayT	-
perihelion	-	DayT	-
countDiff	DayT	Integer	-
$\leq$	DayT	$\mathbb{B}$	-
$>$	DayT	$\mathbb{B}$	-

[The input to the DayDurT could just be DayT, DayT; you don't really need to make the input a tuple. —SS] [Agree —Author]

8.5 Semantics

DayT: natural number

8.5.1 State Variables

stateDay : DayT [You have these same state variables in another module —SS]

8.5.2 Environment Variables

N/A

8.5.3 Assumptions

[It would be neat if you defined a comparison access program to your ADT so that you have greater than implemented. As it is, saying greater than is ambiguous. —SS] [Change the expession of getdiff to countDiff. It uses compare function to check if day1 is greater than day2 —Author]

8.5.4 Access Routine Semantics

new DayT (y, m, d):

- transition: stateDay = $(m = 1 \vee m = 2 \Rightarrow 365.25 \times (y - 1) + 30.6001 \times (m + 13) + d - \text{calculateB}(y/100) + 1720995 \mid \text{True} \Rightarrow 365.25 \times y + 30.6001 \times (m + 1) + d - \text{calculateB}(y / 100) + 1720995)$

  [You don't actually say otherwise using our notation. You could just say True. You also don't have a dummy variable. I don't really know how to read this expression. —SS]

- output: out := self
- exception:

getnext ( ):

- transition:
- output: out := The next day of the stateDay according to the Gregorian Calender.
- exception:

perihelion ( ):

- transition:
- output: out := $(\text{stateDay.m} = 12 \wedge \text{stateDay.d} \geq 21 \Rightarrow (\text{stateDay.y}, 12, 21) \mid \text{True} \Rightarrow (\text{stateDay.y} - 1, 12, 21))$

- exception:

countDiff (day):

- transition:
- output: out := (day $\leq$ stateDay $\Rightarrow$ 1 + countDiff (getnext (day1)) | day < stateDay $\Rightarrow$ 0)
- exception:

$\leq$ (day):

- transition:
- output: out := (day is placed before than stateDay Gregorian Calender $\vee$ inday and day is the same day $\Rightarrow$ True | False)
- exception:

> (day):

- transition:
- output: out := (day is placed before than stateDay Gregorian Calender $\vee$ day and stateDay is the same day $\Rightarrow$ True | False)
- exception:

[You shouldn't have a transition and an output. I don't actually see why you need the daduraL state variable. I would think you could calculate the duration as needed and simply output it? —SS]

8.5.5 Local Functions

calculateB: $\mathbb{N} \rightarrow \mathbb{N}$
calculateB (a) = 2 - a + (a / 4)

9 MIS of Input Parameters Module

9.1 Module

InputPara

9.2 Uses

HardH
Day ADT Module (Section 8)

9.3 Syntax

9.3.1 Exported Constants

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
fromKeyBoard	-	-	Key_error
loadAnaleFile	string	-	File_error
getla	-	DegreeT	-
getph	-	real	-
getpw	-	real	-
getdayS	-	DayT	-
getdayE	-	DayT	-
getdec	-	a sequence of real	-

9.4 Semantics

9.4.1 State Variables

latitude: DegreeT
dayS: DayT [You have these state variables already in other modules. —SS][Delete dayS and dayE in other module —Author]
dayE: DayT
panH: real
panW: real
declination: a sequence [366] of real

9.4.2 Environment Variables

key: Input variables from keyboard.

9.4.3 Assumptions

When users chick the submit bottom from system interface, InputPara.init ( ) implement.

9.4.4 Access Routine Semantics

fromKeyBoard ( ):

- transition: Get the values from users' input.
  latitude, panH, panW = HardH. Φ_P, HardH. P_{A_h}, HardH. P_{A_w}

  dayS := new DayT (HardH. $year_{\text{Start}}$, HardH. $month_{\text{Start}}$, HardH. day_{Start})

dayE := new DayT (HardH. $year_{\mathrm{End}}$, HardH. $month_{\mathrm{End}}$, HardH. day_{End})

- output:
- exception: If the data type of captured values do not match with the parameters' data type ⇒ Key_error

loadAnaleFile (fileName):

- transition: declination [0..366] := read data from the file analemma.txt.
  The text file has the following format, where declination_i denotes the angle of sun declination for day 1 to day 366. All data values are separate into rows, where each row has a value. There is 366 rows in the file.

  declination_0
  declination_1
  declination_2
  .
  .
  .
  declination_366

- output:
- exception: If the file is not found ⇒ File_error

getla ( ):

- transition:
- output: latitude
- exception:

getph ( ):

- transition:
- output: panH
- exception:

getpw ( ):

- transition:
- output: panW

- exception:

getdayS ( ):

- transition:
- output: dayS
- exception:

getdayE ( ):

- transition:
- output: dayE
- exception:

getdec ( ):

- transition:
- output: declination
- exception:

9.4.5 Local Functions

10 MIS of Input Verfication Module

10.1 Module

InputVer

10.2 Uses

InputPara9

10.3 Syntax

10.3.1 Exported Constants

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
verifiedLan	DegreeT	$\mathbb{B}$	-
verifiedP	real, real	$\mathbb{B}$	-
verifiedD	DayT, DayT,	$\mathbb{B}$	-

[Rather than simply rely on exceptions, you could have verify return a Boolean. —SS]

10.4 Semantics

10.4.1 State Variables

10.4.2 Environment Variables

10.4.3 Assumptions

The input values is input from InputPara

10.4.4 Access Routine Semantics

verifiedLan (lan):

- transition:
- output: (lan $>$ 90 $\vee$ lan $<$ - 90 $\Rightarrow$ False | True)
- exception: exc :=

verifiedP (ph, pw):

- transition:
- output: (ph $<$ 0 $\vee$ pw $<$ 0 $\Rightarrow$ False | True)
- exception: exc := if ds and de is a invalid date in Gregorian calendar $\Rightarrow$ invaild_day)

verifiedD (ds, de):

- transition:
- output: (ds $<$ de $\Rightarrow$ False | True)
- exception: exc :=

10.4.5 Local Functions

11 MIS of Output Parameters Module

11.1 Module

OutputPara

11.2 Uses

Table12

11.3 Syntax

11.3.1 Exported Constants

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
addresult	DegreeT, real	-	-
getresult	-	a sequence of a tulple of (DegreeT, real, integer)	-

11.4 Semantics

11.4.1 State Variables

resultAEng: a sequence of a tuple [spell check —SS][yes —Author] of (t: DegreeT, e: real, diff: integer)

11.4.2 Environment Variables

11.4.3 Assumptions

OutputPara.init ( ) always implement before any export access programs in output parameters module. OutputPara.init ( ) will only be called once.

11.4.4 Access Routine Semantics

init ( ):

- transition: resultAEng := $\langle\ \rangle$
- output:
- exception:

addresult (angle, energy):

- transition: resultAEng := resultAEng[0 .. |resultAEng|] $\|$ $\langle$ angle, energy $\rangle$

  [I don't know how to read this. —SS]
- output:
- exception:

getresult ( ):

- transition:

- output: out := resultAEng
- exception:

11.4.5 Local Functions

getdiff : real $\times$ real $\rightarrow$ interger
getdiff (e1, e2) = $\frac{e2}{e1} \times 100$

12 MIS of Table-layout Module

12.1 Module

Table

12.2 Uses

OutputPara11
HardH

12.3 Syntax

12.3.1 Exported Constants

hRow1: "Tilt Angle" hRow2: "Energy Absorption" hRow3: "Energy Comparison"

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
setable	a seqluce of a tuple of (DegreeT, real, integer)	-	-
display	-	-	-

12.4 Semantics

12.4.1 State Variables

resultL: a sequence of a tulple of (t: DegreeT, e: real, diff: integer)

12.4.2 Environment Variables

Output variables will display on the screen. [You should define an environment variable for the screen. —SS]

12.4.3 Assumptions

12.4.4 Access Routine Semantics

setable (result):

- transition: resultL := result
- output:
- exception:

display ( ):

- transition:
- output: out := display a table that shows the result on the screen.

  The text file has the following format, where the first row following heading: hRow1, denotes the result of optimal tilt angle; the second row following heading, hRow2, denotes the result of estimating energy absorption; and the third row following heading, hRow3, denotes the result of different energy absorption between resultL.e [0] and resultL.e [1.. | resultL|] followed by a symbol %

Tilt Angle	resultL.t [0]	...	resultL.t [| resultL|]
Energy Absorption	resultL.e [0]	...	resultL.e [| resultL|]
Energy Competition	resultL.diff [0] %	...	resultL.diff [| resultL|] %

- exception:

12.4.5 Local Functions

13 MIS of Optimum Tilt Angle Module

[Use labels for cross-referencing —SS]

13.1 Module

TitleAng [Short name for the module —SS]

13.2 Uses

SunInten15
ZenithAng16

13.3 Syntax

13.3.1 Exported Constants

$I_S := 1.35$

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
setilt	a sequence of DegreeT	-	-
getilt	-	DegreeT	-
getmaxInten	-	real	out_empty
[accessProg —SS]	-	-	-

13.4 Semantics

13.4.1 State Variables

sunIn: real
zenithL: a sequence of DegreeT
maxsunIn: a tuple of (i: real, d: DegreeT)

[Do you really need state variables? I think you could use an input-output module here. The same comment applies elsewhere. —SS]

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

13.4.2 Environment Variables

N/A [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] setilt(zen):

- transition: sunIn, zenithL, maxsunIn := SunInten.sum (I_S, zen), zen, ⟨ 0, 0 ⟩
- output:
- exception:

[accessProg —SS] getilt ( ):

- transition: maxsunIn := (j: integer | j ∈ [0.. |zenithL|] • j = 0 ⇒ ⟨ SunInten.single (sunIn, zenithL [0]), zenithL [0] ⟩ | ifMax (maxsunIn.i, SunInten.single (sunIn, zenithL [j])) ≠ maxsunIn.i ⇒ ⟨ SunInten.single (sunIn, zenithL [j]), zenithL [j]⟩) [This is a complicated expression. I think you aren't quite following our notation. Also, for a complicated expression like this, it helps to add local functions. —SS]
- output: out := maxsunIn.d
- exception:

getmaxInten ( ):

- transition:
- output: SunInten.single (sunIn, maxsunIn.d)
- exception: exp := maxsunIn.i = 0 ∧ maxsunIn.d ⇒ out_empty

[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

ifMax: real × real → real
ifMax(x, y) = (x ≥ y ⇒ x | otherwise ⇒ y)

[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]

14 MIS of Solar Energy Absorption Module

14.1 Module

Energy

14.2 Uses

InputPara9
ZenithAng16
TiltAng13

14.3 Syntax

14.3.1 Exported Constants

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
setenergy	real, real, DegreeT, a sequence of DegreeT	-	-
getavenergy	-	real	-

14.4 Semantics

14.4.1 State Variables

energyL: a sequence of real

14.4.2 Environment Variables

N/A

14.4.3 Assumptions

ZenithAngle has been fully implement before implement Energy.init ( ).
Energy.init ( ) is called first before any other exported access programs.

14.4.4 Access Routine Semantics

setenergy (pw, ph, maxinten, zen):

- transition: energyL := getenergy (getsunIn (zen, maxinten),pw, ph)

- output:
- exception:

getavenergy ( ):

- transition:
- output: out := + (i: integer | i ∈ [0.. |energyL|] • energyL[i]) / |energyL|
- exception:

14.4.5 Local Functions

getsunIn: a sequence of DegreeT → a sequence of real
getsunIn (z, maxinten) = || (i: integer | i ∈ [0.. |z|] • [SunInten.single(maxinten, z[i])])

getenergy: a sequence of real × real × real → a sequence of real
getenergy (e, pw, ph) = || (i: integer | i ∈ [0.. |e|] • [pw × ph × 18.7 × 0.75 × e [i]])

15 MIS of Sun Intensity Equation Module

[Use labels for cross-referencing —SS]
[You can reference SRS labels, such as R??. —SS]
[It is also possible to use LaTeXfor hyperlinks to external documents. —SS]

15.1 Module

SunInten

15.2 Uses

ZenithAngle16

15.3 Syntax

15.3.1 Exported Constants

$I_S := 1.35$

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
sum	real, a sequence of DegreeT	real	sequence_empty
single	real, DegreeT	real	-

15.4 Semantics

15.4.1 State Variables

15.4.2 Environment Variables

N/A

15.4.3 Assumptions

15.4.4 Access Routine Semantics

sum (e, z):

- output: out := + (i: integer | i ∈ [0.. |z|] •

$$I_S \cdot (\frac{1.00}{e})^{sec(z[i])})$$

- exception: exc := |z| = 0 ⇒ sequence_empty

single (e, z):

- output: out :=

$$I_S \cdot (\frac{1.00}{e})^{sec(z)}$$

- exception:

15.4.5 Local Functions

16 MIS of Zenith Angle Equation Module

16.1 Module

ZenithAng

16.2 Uses

DayDurTy8
SunCatTy7

16.3 Syntax

16.3.1 Exported Constants

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
setzen	a sequence of AnalammaT, a sequence of Integer	-	-
getzen	real	a sequence of DegreeT	sequence_empty

16.4 Semantics

16.4.1 State Variables

decS: a sequence of DegreeT

[As I mentioned earlier, I don't think you need state variables. An input-output relation should work fine. —SS]

16.4.2 Environment Variables

16.4.3 Assumptions

Read File Module will always be implemented before Zenith angle Module. [Do you mean "called" or "invoked" rather than "implemented"? —SS]
InputPara Module will always be implemented before Zenith angle Module.
ZenithAng.setzen is called before any other access program.
ZenithAng.setzen is called by the Read File Module.

16.4.4 Access Routine Semantics

setzen (a, durlist):

- transition: decS := (i: integer | i $\in$ [0.. |durlist|] $\bullet$ decS $\|$ $\langle$ getdec (a, durlist[i])$\rangle$)
- output:
- exception:

getzen (lan):

- transition:
- output: out := out $\|$ (i: integer | i $\in$ [0.. |decS|] $\bullet$ lan $\times$ decS [i] $\geq 0 \Rightarrow$ $\langle$ lan - decS [i] $\rangle$ | lan $\times$ decS [i] $< 0 \Rightarrow$ $\langle$ lan + decS [i] $\rangle$)
- exception: |decS| $= 0 \Rightarrow$ sequence_empty

16.4.5 Local Functions

getdec : a sequence of AnalammaT × integer → DegreeT
getdec (a, d) :=

$$\arcsin \frac{a.z[d]}{\sqrt{a.x^2[d]+a.y^2[d]+a.z^2[d]}}$$

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.

[Your MIS seems more complicated than it has to be. The purpose of all the state variables is not clear. I suggest you review all of the modules to see if they are necessary and whether you really need the state variables. You also probably do not need so many modules. You could combine the modules that export calculations into a “Calculation” module. Also, you might want to explain what is going on in words, in case your math doesn’t say what you think it is saying. —SS]

17 Appendix

[Extra information if required —SS]