Module Interface Specification for Sun Catcher

Sharon (Yu-Shiuan) Wu

December 26, 2019

1 Revision History

Date	Version	Notes
2019/11/25	1.0	First Version
2019/12/19	1.2	Second Version

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 **?**, with the addition that template modules have been adapted from **?**. The mathematical notation comes from Chapter 3 of **?**. 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
sequence	[ ]	a sequence of the same type

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 Modules	
Behaviour-Hiding Module	Control Module Input/Read Parameters Module Input Verification Module Solar Energy Absorption Module Optimum Tilt Angle Module Calculation 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

SunCatTy (Section 7)
DayT (Section 8)
InputPara (Section 9)
[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)
TiltAng (Section 12)
Energy (Section 13)
Calculation (Section 14)

6.3 Syntax

6.3.1 Exported Constants

filename = ”analemma.txt”

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
input	-	-	-
getOneAng [main? —SS][changed name —Author]	-	-	-
getTwoAng	-	-	-
output	-	-	-

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 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] [Yes, Agree. But I will still keep some of my state variable inside the document for the purpose the easy understanding. Then implement in a slightly different way. —Author]

6.4.2 Environment Variables

6.4.3 Assumptions

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

6.4.4 Access Routine Semantics

input ( ):

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

  InputPara.load_anale_data (filename)

  # Verify the input values using Input Verfication Module
  verifiedLat (InputPara.getla ( ))
  verifiedP (InputPara.getph ( ), InputPara.getpw ( ))
  verifiedD (InputPara.getdayS, InputPara.getdayE)

  # If verifiedLat, verifiedP, and verifiedD are all true
  intermidiate (InputPara.getdec ( ), InputPara.getdayS ( ), InputPara.getdayE ( ), InputPara.getpw ( ) , InputPara.getph ( ) , InputPara.getla ( ))

intermidiate (dec [ ], dayS, dayE, pw, ph, latitude):

- transition:

  zenith = zenithList (dec [ ], dayS, dayE)

  cutList = makeCut (2, | zenith |)

  angle = $\|$ ((a, b) : ($\mathbb{N}$, $\mathbb{N}$) | (a, b) $\in$ cutList $\bullet$ optAngle (zenith [a*b...(a * b + b)], latitude))

  cupZipAng = zipList(cutList, angle)

energy = $\|$ (((a, b), ang) : ((ℕ, ℕ), real) | ((a, b), ang) ∈ cupZipAng • optEnergy (zenith [a*b...(a * b + b)], ang, pw, ph))

day = getDayList (cutList, dayS)

addResult (angle, energy, day)

optEnergy (zen [ ], tiltAngle, pw, ph):
# Get the estimated solar absorption using Optimum Tilt Angle Module

- output: out :=

  getEnergy (InputPara.getpw, InputPara.getph, localtiltSunIn (localsunIn ($\theta_{S_{\text{date}}}$, θ_T), $\theta_{S_{\text{date}}}$))

optAngle (zen [ ], latutude):
# Get the optimum tilt angle using $\theta_{S_{\text{date}}}$

- output: out := getilt ($\theta_{S_{\text{date}}}$, localsunIn($\theta_{S_{\text{date}}}$))

makeCut (cut, len):

- output: out := $\|$ (i : ℕ | i ∈ cut • localCut (i, listLen))

zenithList (dec [ ], dayS, dayE):

- output: out := Calculation.getzenList (localScaleDec(dec,dayS,dayE) localDaySandPer ( ), localDaySandDayE ( ) , InputPara.getlala)

6.4.5 Local Functions

# The days detween start day and perhelion
localDaySandPer: Integer
localDaySandPer = InputPara.getdayS. perhelion

# The days detween start day and end day
localDaySandDayE: Integer
localDaySandDayE =InputPara.getdayE.countDiff (InputPara.getdayS)

# Calculate the total sun intensity for the base case
localsunIn: a sequence of DegreeT $\rightarrow$ real
localsunIn (zen[ ]) = Calculation.sumSunIn(1.35, zen)

# Get the optimum sun intensity
localtiltSunIn: DegreeT $\times$ real $\rightarrow$ real
localtiltSunIn (degree, intensity) = TiltAngle.getiltInten (intensity, degree)
localScaleDec : DegreeT [ ] $\times$ DayT $\times$ DayT $\rightarrow$ DegreeT [ ] localScaleDec (dec, ds, ds)
[It is nice to have a newpage between modules. —SS][OK —Author]

localCut : $\mathbb{N} \times \mathbb{N} \rightarrow$ a sequence of tuple ($\mathbb{N}$, $\mathbb{R}^+$)
localCut (i, lenght) = $\|$ (x : $\mathbb{N}$ | x $\in$ i $\bullet$ (x-1, lenght / i))

zipList : a sequence $\times$ a sequence $\rightarrow$ a sequence
zipList ([a] , [b]) = $\|$ (x : $\mathbb{N}$ | x $\in$ | a | $\bullet$ (a[x], b[x]))

7 Sun Catcher Type Module

7.1 Module

SunCatTy

7.2 Uses

N/A

7.3 Exported Types

DegreeT = $\mathbb{R}$

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

N/A

8 Day ADT Module

8.1 Template Module

DayT

8.2 Uses

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
addDay	$\mathbb{N}$	DayT	-
perihelion	-	Integer	-
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 $\Rightarrow$ MonthOneTwo (y, m, d) | True $\Rightarrow$ MonthNotOneTwo (y, m, d)

  [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:

addDay (times):

- transition:
- output: out := (times $\neq$ 0 $\Rightarrow$ localnextday (stateDay).addDay(times -1) | True $\Rightarrow$ stateDay)
- exception:

perihelion ( ):

- transition:
- output: out := (stateDay.m = 12 $\wedge$ stateDay.d $\geq$ 21 $\Rightarrow$ countDiff (new DayT (stateDay.y, 12, 21)) | True $\Rightarrow$ countDiff (new DayT (stateDay.y - 1, 12, 21)))
- exception:

countDiff (day):

- transition:
- output: out := (day $\leq$ stateDay $\Rightarrow$ 1 + countDiff (localnextday (day)) | 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)

MonthOneTwo: $\mathbb{N} \times \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$
MonthOneorTwo (y, m, d) = 365.25 $\times$ (y - 1) + 30.6001 $\times$ (m + 13) + d - calculateB(y /100) + 1720995

MonthNotOneTwo: $\mathbb{N} \times \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$
MonthNotOneorTwo (y, m, d) = 365.25 $\times$ y + 30.6001 $\times$ (m + 1) + d - calculateB(y / 100) + 1720995)

localnextday : DayT $\rightarrow$ DayT
localnextday (inputDay) = The next day of the inputDay according to the Gregorian Calender.

9 MIS of Input Parameters Module

9.1 Module

InputPara

9.2 Uses

HardH
Day ADT Module (Section 8) SunCatTy (Section 7)

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 = key. Φ_P, key. P_{A_h}, key. P_{A_w}

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

  dayE := new DayT (key. $year_\text{End}$, key. $month_\text{End}$, key. 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

InputPara (Section 9) SunCatTy (Section 7)
DayT (Section 8)

10.3 Syntax

10.3.1 Exported Constants

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
verifiedLat	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

# Display warning message on the screen
screen: Hardware.screen

10.4.3 Assumptions

The input values is input from InputPara

10.4.4 Access Routine Semantics

verifiedLat (latitude):

- transition:
- output: latitude $\leq 90 \wedge$ atitude $\geq$ - 90

- exception: exc :=

verifiedP (ph, pw):

- transition:
- output: $ph > 0 \wedge pw > 0$
- exception: exc :=

verifiedD (ds, de):

- transition:
- output: $ds < de$
- exception: exc :=

10.4.5 Local Functions

11 MIS of Table-layout Module

11.1 Module

Table

11.2 Uses

HardH
SunCatTy (Section 7)
DayT (Section 8)

11.3 Syntax

11.3.1 Exported Constants

mainColumn1: "Adjust How Many Times"
mainColumn2: "Energy Absorption/day"
angleColumn1: "Optimum Angle"
angleColumn2: "Optimum Angle"
fileForMian: "MainTable.txt" fileForAngle: "AngleTable.txt"

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
addresult	a set of DegreeT, a set of real, a set of DayT	-	-
display	-	-	-

11.4 Semantics

11.4.1 State Variables

mainTable: a set of { tuples of (cutTime: natural number, energy: real) }
tiltTable: a set of { tuples of (time: a set of DayT, angle: a set of DegreeT) }

11.4.2 Environment Variables

writeFile: Write tables (output variables) in the file. [You should define an environment variable for the screen. —SS]

11.4.3 Assumptions

11.4.4 Access Routine Semantics

addresult (angle, energy, day):

- transition: mainTable = mainTable $\cup$ { (|angle| , energy) }
  tiltTable = tiltTable $\cup$ { (day, angle) } [I don't know how to read this. —SS] [Change the expression —Author]
- output:
- exception:

display ( ):

- transition:
- output: out := display a table that shows the result in the file.

  The main table has the following format, where the first row following heading: mainColumn1, denotes the time of adjust the solar panel; the second row following heading, mainColumn2, denotes the result of estimating energy absorption per day

  mainSet: a set of tuples | mainSet $\in$ mainTable $\wedge$ main: (cutTime: natural number, energy: real) | main $\in$ mainSet

Adjust How Many Times	Energy Absorption/day
main.cutTime	main.energy

  writeFile.(fileForMain)

  The angle table has following format, where the fist column heading: angleColumn1, denotes the time for adjust the solar panel; and the second column heading: angleColumn2, denotes the angle for adjusting the solar panel; the row heading denotes the time for adjusting the angle.

  tiltSet: a set of tuples | tiltSet $\in$ tiltTable $\wedge$ tilt: (time: a set of DayT, angle: a set of DegreeT) | tilt $\in$ tiltSet

Time	Optimum Angle
tilt.time	tilt.angle

  writeFile.(fileForAngle)
- exception:

11.4.5 Local Functions

12 MIS of Optimum Tilt Angle Module

[Use labels for cross-referencing —SS]

12.1 Module

TitleAng [Short name for the module —SS]

12.2 Uses

Calculation (Section 14)
SunCatTy (Section 7)

12.3 Syntax

12.3.1 Exported Constants

$I_S := 1.35$

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
getilt	DegreeT[ ], real	DegreeT	-
getiltInten	DegreeT, real	real	-
[accessProg —SS]	-	-	-

12.4 Semantics

12.4.1 State Variables

tiltDegree: 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]

12.4.2 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]

12.4.3 Assumptions

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

12.4.4 Access Routine Semantics

[accessProg —SS] getilt (zenithL[ ], sunIn):

- transition:
- output: out := tiltDegree, such that
  $\forall$(zen: DegreeT | j $\in$ zenithL $\bullet$ tiltDegree = localMax (tiltDegree, zen, sunIn) [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]
- exception:

getiltInten (degree, sunIn):

- transition:
- output: Calculation.sglSunIn (degree, sunIn)
- exception:

[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]

12.4.5 Local Functions

# Get two tuples, then return a tuple.
localMax: DegreeT $\times$ DegreeT $\times$ real $\rightarrow$ DegreeT
localMax(origDegree, newDegree, sunIn) = (getiltInten (origDegree, sunIn) $\geq$ getiltInten (newDegree, sunIn) $\Rightarrow$ origDegree | True $\Rightarrow$ newDegree)

[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]

13 MIS of Solar Energy Absorption Module

13.1 Module

Energy

13.2 Uses

InputPara (Section 9)
Calculation (Section 14)
TiltAng (Section 12) SunCatTy (Section 7)

13.3 Syntax

13.3.1 Exported Constants

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
getEnergy	real, real, real, DegreeT [ ]	real	-

13.4 Semantics

13.4.1 State Variables

energyL: a sequence of real

13.4.2 Environment Variables

N/A

13.4.3 Assumptions

13.4.4 Access Routine Semantics

getEnergy (pw, ph, maxInten, zen[ ]):

- transition:
- output: localAverage (localEnergy (localSunIn (zen, maxInten),pw, ph))
- exception:

13.4.5 Local Functions

localSunIn: a sequence of DegreeT $\times$ real $\rightarrow$ a set of real
localSunIn (zen[ ], maxinten) = $\cup$ (i: DegreeT | i $\in$ z $\bullet$ { SunInten.single(maxinten, i) })

localEnergy: a set of real $\times$ real $\times$ real $\rightarrow$ a set of real
localEnergy (sunIn, pw, ph) = $\cup$ (i: integer | i $\in$ sunIn $\bullet$ { pw $\times$ ph $\times$ 18.7 $\times$ 0.75 $\times$ i })

localAverage : a set of real $\rightarrow$ real
localAverage (energy) = + (i: real | i $\in$ energy $\bullet$ i) / |energy|

14 MIS of Calculation

14.1 Module

Calculation

14.2 Uses

SunCatTy (Section 7)

14.3 Syntax

14.3.1 Exported Constants

$I_S := 1.35$

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
getzenList	DegreeT[ ], integer, integer	-	-
sumSunIn	DegreeT[ ], real	real	-
sglSunIn	DegreeT, real	real	-

14.4 Semantics

14.4.1 State Variables

14.4.2 Environment Variables

14.4.3 Assumptions

14.4.4 Access Routine Semantics

getzenList(decList[ ], i , diff, latitude):
# Calculate every elemant in list, decList[i.. i+diff]. Then output the list

- transition:
- output: $\|$ (dec: DegreeT $|$ dec $\in$ decList[i.. i+diff] $\bullet$ localZenAngle (dec, latitude)
- exception:

getDayList (cut [ ], dayS):

- transition:
- output: ‖ ((a, b) : (ℕ, ℕ) | (a,b) ∈ cut • dayS.addDay (a * b))
- exception:

sumSunIn (zenList[ ], energy):

- output: out := + (zen: DegreeT | zen ∈ zenList •

$$I_S \cdot (\frac{1.00}{\text{energy}})^{sec(\text{zen})})$$

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

sglSunIn (zen, energy):

- output: out :=

$$I_S \cdot (\frac{1.00}{\text{energy}})^{sec(zen)}$$

- exception:

14.4.5 Local Functions

localZenAngle: DegreeT × real → Degree
localZenAngle (dec, latitude) = (dec * latitude < 0 ⇒ dec + latitude | True ⇒ dec - latitude)

[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]

15 Appendix

[Extra information if required —SS]