Module Interface Specification for SynthEddy

Phil Du (Software)
Nikita Holyev (Theory)

March 18, 2024

1 Revision History

Date	Version	Notes
2024-03-18	1.0	Initial MIS

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at GitHub repo

Contents

3 Introduction

The following document details the Module Interface Specifications for SynthEddy, a software to artificially generate flow field that mimics turbulent flow, which can be use as starting point for CFD simulation.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at SRS, MG.

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 SynthEddy.

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 (-∞, ∞)
string	str	an array of char

The specification of SynthEddy 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, SynthEddy 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	Query Interface Eddy Profile Module Flow Field Module Eddy Module Shape Function Module Vector Module
Software Decision Module	Main Control Module File I/O Module Visualization Module

Table 1: Module Hierarchy

6 MIS of Query Interface

6.1 Module

`query`

6.2 Uses

- Parse query input to call flow field module, serialize and return the result
- Queue multiple queries

6.3 Syntax

6.3.1 Exported Constants

None.

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
`handle_request`	`request` str	`response` str	`InvalidRequestFormat`

6.4 Semantics

6.4.1 State Variables

None.

6.4.2 Environment Variables

None.

6.4.3 Assumptions

None.

6.4.4 Access Routine Semantics

`handle_request(request)`:

- output: out := `encode({VectorT})`, velocity vectors at all the queried points in one JSON string.

- exception: exc := (`decode(request)` is not {VectorT} $\Rightarrow$ `InvalidRequestFormat`), the request should be a JSON string of positions vectors to query get the velocity vectors.

More request methods to be implemented in the future.

6.4.5 Local Functions

`decode(request)`:

- output: out := {VectorT}, an array of position vectors.

7 MIS of Eddy Profile Module

7.1 Module

eddy_profile

7.2 Uses

- (Currently) Load eddy profile defined by the user [SRS: A3].
- (Future) Generate eddy profile based on user input parameters [MG: AC1].

7.3 Syntax

7.3.1 Exported Constants

None.

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
load	profile_name str	-	InvalidProfile
get_params	-	params $\{\{\mathbb{R}+, \mathbb{R}+\}\}$	-
get_weights	-	weights $\{\mathbb{R}+\}$	-

7.4 Semantics

7.4.1 State Variables

- name: str, name of the eddy profile (is also filename).
- params: $\{\{\mathbb{R}+, \mathbb{R}+\}\}$, array of strength and radius sets for each type of eddy.
- weights: $\{\mathbb{R}+\}$, corresponding array of weights for each type of eddy.

7.4.2 Environment Variables

None.

7.4.3 Assumptions

None.

7.4.4 Access Routine Semantics

`load(profile_name)`:

- transition: `params, weights` := `fileIO.read('profile', profile_name)`
- exception: exc := (`params` is not $\{\{\mathbb{R}+, \mathbb{R}+\}\}$ or `weights` is not type $\{\mathbb{R}+\}$ $\Rightarrow$ `InvalidProfile`)

`get_params()`:

- output: out := `params`

`get_weights()`:

- output: out := `weights`

7.4.5 Local Functions

Some functions to compute the eddy profile based on user provided parameters, to be implemented in the future.

8 MIS of Flow Field Module

8.1 Module

`flow_field`

8.2 Uses

- Create flow field based on eddy profile or load existing one.
- Compute velocity vector at given position and time.

8.3 Syntax

8.3.1 Exported Constants

None.

8.3.2 Exported Access Programs

Name	In	Out	Exceptions

`init`	`profile` EddyProfileT,
	`field_name` str,
	`dimensions` VectorT,
	`avg_vel` $\mathbb{R}$,
	`eddy_count` $\mathbb{Z}$
		-	`InvalidDimensions`
			`InvalidAvgVelocity`
			`EddyScaleTooLarge`

`load`	`field_name` str	-	-
`save`	-	`field` Record	-
`get_vel`	`position` VectorT, `time` $\mathbb{R}$	`velocity` VectorT	`OutOfBoundary` `NegativeTime`

8.4 Semantics

8.4.1 State Variables

- `profile`: eddy_profileT, eddy profile to be used to generate the flow field.
- `name`: str, name of the flow field (is also filename).
- `dimensions`: Vector T of $\{L_x, L_y, L_z\}$, size of the flow field, with x being the axial direction, y vertical and z horizontal.
- `avg_vel`: $\mathbb{R}$, average flow velocity along x-axis.
- `eddies`: {EddyT}, array of eddies in the flow field.

8.4.2 Environment Variables

None.

8.4.3 Assumptions

- External flow [SRS: A4, MG: AC2]

8.4.4 Access Routine Semantics

`init(profile, field_name, dimensions, avg_vel, eddy_count)`:

- transition:
  - `profile, name, length_x, length_y, length_z, avg_vel` := as inputted
  - `eddies` := {`eddy_count` number of EddyT randomly generated with `profile`}
- exception:
  - exc := (any of d in `dimensions` $\leq 0 \Rightarrow$ `InvalidDimensions`)
  - exc := (`avg_vel` $< 0 \Rightarrow$ `InvalidAvgVelocity`)
  - exc := (any of 2∗{radius in `profile`} $\geq$ any of d in `dimensions` $\Rightarrow$ `EddyScaleTooLarge`)

`load(field_name)`:

- transition: all state variables reconstructed from `fileIO.read('field', field_name)`.

`save(new_name?)`:

- transition: `name` := `new_name` if provided, otherwise name remains the same.
- output: out := Record (dict) with all current state variables. Each EddyT in `eddies` is also converted to a Record. So that `fileIO.write('field', Record)` can be called to serialized the current state of the flow field into a JSON string for saving.

`get_vel(position, time)`:

- output: out := $\sum$ {`eddy.get_vel(position - eddy_pos(eddy, time))` for each `eddy` in `eddies`}, with wrap-around applied for eddies near the boundary.

8.4.5 Local Functions

`eddy_pos(eddy, time)`:

- output: out := VectorT: {
  `get_offset(time) + eddy.get_init_x(),`
  `eddy.get_y(get_iter(time), dimensions[1]),`
  `eddy.get_z(get_iter(time), dimensions[2])`
  }the center position of the eddy at a given time (moving due to flow).

`get_iter(time)`:

- output: out := $\texttt{round(}\frac{\texttt{avg\_vel*time}}{\texttt{dimensions[0]}}\texttt{)}$, the number of iterations of the flow field at a given time (how many x-lengths have passed due to average flow velocity).

`get_offset(time)`:

- output: out := $\texttt{avg\_vel} * \texttt{time}\%\texttt{dimensions[0]}$, the x-offset of the flow field at given time in current iteration.

9 MIS of Eddy Module

9.1 Module

`eddy`

9.2 Uses

- Create eddy object based on parameters given.
- Compute velocity vector relative position from center.

9.3 Syntax

9.3.1 Exported Constants

None.

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
`init`	`field_dimensions` VectorT, `strength` $\mathbb{R}+$, `radius` $\mathbb{R}+$, `orient` VectorT	-	-
`get_init_x`	-	`init_x` $\mathbb{R}$	-
`get_y`	`iter` $\mathbb{Z}$, `length_y` $\mathbb{R}+$	`y` $\mathbb{R}$	-
`get_z`	`iter` $\mathbb{Z}$, `length_z` $\mathbb{R}+$	`z` $\mathbb{R}$	-
`get_vel`	`rel_position` VectorT	`velocity` VectorT	-

9.4 Semantics

9.4.1 State Variables

- `init_x`: $\mathbb{R}$, the initial offset from zero x-position.
- `strength`: $\mathbb{R}+$, intensity of the eddy.
- `radius`: $\mathbb{R}+$ for length scale. Velocity outside of `radius` is always zero.
- `orient`: VectorT, unit vector describing the orientation of the eddy spin axis.
- `y_arr`: $\{\mathbb{R}\}$, array of y-positions at each flow iteration.

- `z_arr`: $\{\mathbb{R}\}$, array of z-positions at each flow iteration.

9.4.2 Environment Variables

None.

9.4.3 Assumptions

- EddyT objects are not created manually, but generated by calls from `flow_field`. Thus, the caller has ensured the validity of all parameters.

9.4.4 Access Routine Semantics

`init(init_x, strength, radius, orient)`:

- transition:
  - `strength, radius, orient` := as inputted
  - `init_x` := `rand(0, field_dimensions[0])`
  - `y[0], y[1], y[2]` := {`rand(0, field_dimensions[1])`}
  - `z[0], z[1], z[2]` := {`rand(0, field_dimensions[2])`}

`get_init_x()`:

- output: out := `init_x`

`get_y(iter, length_y)`:

- transition: `y[iter]` := `rand(0, length_y)` if `y[iter]` does not exist.
- output: out :=`y[iter]`

`get_z(iter, length_z)`:

- transition: `z[iter]` := `rand(0, length_z)` if `z[iter]` does not exist.
- output : out :=`z[iter]`

`get_vel(rel_position)`:

- output: out := computed from `rel_position` with `strength, radius, orient` and `ShapeFunction.active(rel_position, radius)`, see [SRS: TM1, TM2]

`dump()`:

- output: out := Record of all current state variables. Used by `flow_field.save()` to serialize the eddy object.

9.4.5 Local Functions

None.

10 MIS of Shape Function Module

10.1 Module

`shape_function`

10.2 Uses

- A library of shape functions to be used by `eddy` (9).

10.3 Syntax

10.3.1 Exported Constants

None.

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
`set_active`	`active_func` Function	-	-
`active`	`rel_position` VectorT, `radius` $\mathbb{R}$	`shape_val` $\mathbb{R}$	-
`squared`	`rel_position` VectorT, `radius` $\mathbb{R}$	`shape_val` $\mathbb{R}$	-
`gaussian`	`rel_position` VectorT, `radius` $\mathbb{R}$	`shape_val` $\mathbb{R}$	-
`...`	`rel_position` VectorT, `radius` $\mathbb{R}$	`shape_val` $\mathbb{R}$	-

User can modify this module to add more shape functions.

10.4 Semantics

10.4.1 State Variables

- `active`: The function that is currently designated as the active shape function.

10.4.2 Environment Variables

None.

10.4.3 Assumptions

None.

10.4.4 Access Routine Semantics

`set_active(active_func)`:

- transition: `active` := `active_func`,
  so that other modules can always call `shape_function.active()` to use the designated function. This should be set in `main` when the program starts.

`active()`:

- output: out := `init_x`

`active(rel_position, radius)`:

- output: out := shape function value, depending on the active shape function.

`squared(rel_position, radius)`:

- output: out := shape function value computed by taking the distance from the rel_position to the center `mag(rel_position)`, and `radius` (or length sclae) of the eddy. See [SRS: TM1].

`gaussian(rel_position, radius)`:

- output: out := Use a different (gaussian) equation to get the above value, as may be preferred by some researchers.

10.4.5 Local Functions

None.

11 MIS of Main Control Module

11.1 Module

`main`

11.2 Uses

- Take command line arguments and call various modules accordingly.

11.3 Syntax

11.3.1 Exported Constants

None.

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
`main`	`command_args` str,	-	-

11.4 Semantics

11.4.1 State Variables

None.

11.4.2 Environment Variables

- Command line console.

11.4.3 Assumptions

None.

11.4.4 Access Routine Semantics

`main(command_args)`:

- `--new_field --name <field_name> --dim <Lx>,<Ly>,<Lz>, --vel <avg_vel> --count <eddy_count> --profile <profile_name>`
  - call `eddy_profile.load(profile_name)` to read the eddy profile file.
  - call `flow_field.init(profile, field_name, dimensions, avg_vel, eddy_count)` to create a new flow field.

- `--field <field_name> --query <request>`
  - call `flow_field.load(field_name)` to load the flow field.
  - call `query.handle_request(request)` to get the velocity vectors at the queried points.

11.4.5 Local Functions

None.

12 MIS of File I/O Module

12.1 Module

`fileIO`

12.2 Uses

- Read and write JSON files for eddy profile and flow field.

12.3 Syntax

12.3.1 Exported Constants

None.

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
`read`	`type` str, `name` str	Record or Array	FileNotExist
`write`	`type` str, `content` Record or Array	-	FailToWrite

12.4 Semantics

12.4.1 State Variables

None.

12.4.2 Environment Variables

- Files on disk.

12.4.3 Assumptions

- The field name or profile name is the same as the filename.
- Saved fields are in `./fields/` and saved profiles are in `./profiles/` directories

12.4.4 Access Routine Semantics

`read(type, name)`:

- output: out := Record or Array, the parsed content of the file.
- exception: exc := (file cannot be found at `./<type>/<name>.json` ⇒ FileNotExist)

`write(type, content)`:

- transition: write the serialized JSON string to the file on disk.
- exception: exc := (file cannot be written to disk ⇒ FailToWrite)

12.4.5 Local Functions

None.

13 MIS of Visualization Module

THIS IS A PLACEHOLDER [MG: AC5]

13.1 Module

`visualize`

13.2 Uses

- Render 2D or 3D images of the flow field.

13.3 Syntax

13.3.1 Exported Constants

None.

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
...	...	...	...

13.4 Semantics

13.4.1 State Variables

?

13.4.2 Environment Variables

?

13.4.3 Assumptions

?

13.4.4 Access Routine Semantics

?

13.4.5 Local Functions

?

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.

14 Appendix

[Extra information if required —SS]

15 Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design. Please answer the following questions:

1. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)
2. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)