Module Interface Specification for Kaplan

Jen Garner

November 25, 2018

1 Revision History

Date	Version	Notes
November 20, 2018 (Tuesday)	1.0	Initial draft

2 Symbols, Abbreviations and Acronyms

See https://github.com/PeaWagon/Kaplan/blob/master/docs/SRS/SRS.pdf Documentation.
cid = compound identification number (for https://pubchem.ncbi.nlm.nih.gov/ website)
vetee = private database repository on github

Contents

3 Introduction

The following document details the Module Interface Specifications for Kaplan. This program is designed to search a potential energy space for a set of conformers for a given input molecule. The energy and RMSD are used to optimize dihedral angles, which can then be combined with an original geometry specification to determine an overall structure for a conformational isomer.

Complementary documents include the System Requirement Specifications (SRS) and Module Guide (MG). The full documentation and implementation can be found at `https://github.com/PeaWagon/Kaplan`.

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 $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

Jen is also using this link for now. Also, the PEP8 style guide from Python will be used for naming conventions.

The following table summarizes the primitive data types used by Kaplan.

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 (-∞, ∞)
boolean	bool	True or False

The specification of Kaplan uses some derived data types: sequences, strings, tuples, lists, and dictionaries. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a fixed list of values, potentially of different types. Lists are similar to tuples, except that they can change in size and their entries can be modified. For strings, lists, and tuples, the index can be used to retrieve a value at a certain location. Indexing starts at 0 and continues until the length of the item minus one (example: for a list my_list = [1,2,3], my_list[1] returns 2). A slice of these data types affords a subsection of the original data (example: given a string s = "kaplan", s[2:4] gives "pl"). Notice that the slice's second value is a non-inclusive bound. A dictionary is a dynamic set of key-value pairs, where the keys and the values can be modified and of any type. A dictionary value is accessed by calling its key, as in dictionary_name[key_name] = value.

Kaplan uses three special objects called Pmem, Ring, and Parser. These objects have methods that are described in 13, 12, and 15 respectively. The Python NoneType type object is also used.

Here is a table to summarize the derived data types:

Data Type	**Notation**	**Description**
population member	Pmem	an object used by Kaplan to represent potential solutions to the conformer search/optimization problem
ring	Ring	an object used by Kaplan to store Pmem objects and define how they are removed, added, and updated
parser	Parser	a Vetee object used by Kaplan to represent the molecular geometry, its energy calculations, and input/output parameters; also inherited classes include: Xyz, Com, Glog, Structure
string	str	a string is a list of characters (as implemented in Python)
list	list or [ ]	a Python list (doubly-linked)
dictionary	dict or {}	a Python dictionary that has key value pairs
NoneType	None	empty data type

In addition, Kaplan 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	GA Input Molecule Input GA Control Fit_G Tournament Crossover & Mutation Ring Pmem Output
Software Decision Module	Geometry Energies RMSD

Table 1: Module Hierarchy

6 MIS of GA Input

[You can reference SRS labels, such as R1. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

The purpose of this module is to provide a utility for reading and verifying input related to the genetic algorithm (GA). There are two main functions: read_ga_input and verify_ga_input. The first function opens a data file (.txt file) with the following format:

```
num_mevs = 1000
num_slots = 100
num_filled = 20
num_geoms = 3
num_atoms = 10
t_size = 7
num_muts = 3
num_swaps = 1
pmem_dist = 5
fit_form = 0
coef_energy = 0.5
coef_rmsd = 0.5
```

These values are read into a Python dictionary, called ga_input_dict. The order of the inputs does not matter, but Kaplan will throw an error if one of the keys is missing. This dictionary is then passed to the second function, which checks that the values are correct and that all keys have been given. From the SRS document, n_G and n_a are represented here as num_geoms and num_atoms respectively. Also, coef_energy and coef_rmsd are C_{E} and C_{RMSD} from the SRS. All keys are case insensitive.

[I may have to check if SMILES strings are case sensitive for the programs I am parsing them with. —JG]

6.1 Module

ga_input

6.2 Uses

None

6.3 Syntax

6.3.1 Exported Constants

NUM_GA_ARGS := 12

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
read_ga_input	str	dict	FileNotFoundError
verify_ga_input	dict	None	ValueError

6.4 Semantics

6.4.1 State Variables

num_args := $+(x : list | key \in x \wedge value \in x \wedge length(x) = 2 : 1)$

ga_input_dict, which is a dictionary that contains:

- num_mevs := $\mathbb{N}$
- num_geoms := $\mathbb{N}$
- num_atoms := $\{x \in \mathbb{N} : x > 3\}$
- num_slots := $\{x \in \mathbb{N} : x \geq \text{num\_filled}\}$
- num_filled := $\{x \in \mathbb{N} : x \leq \text{num\_slots}\}$
- num_muts := $\{0 \vee x \in \mathbb{N} : \text{num\_atoms} \geq x \geq 0\}$
- num_swaps := $\{0 \vee x \in \mathbb{N} : \text{num\_geoms} \geq x \geq 0\}$
- t_size := $\{x \in \mathbb{N} : 2 \geq x \leq \text{num\_filled}\}$
- pmem_dist := $\{0 \vee x \in \mathbb{N} : x \geq 0\}$
- fit_form := $\{0 \vee x \in \mathbb{N} : x \geq 0\}$
- coef_energy: $\mathbb{R}$
- coef_rmsd: $\mathbb{R}$

6.4.2 Environment Variables

ga_input_file: str representing the file that exists in the working directory (optionally includes a prepended path).

6.4.3 Assumptions

This module is responsible for all type checking and no errors will come from incorrect passing of variables (except input related to the molecule - covered in 7).

6.4.4 Access Routine Semantics

read_ga_input(ga_input_file):

- transition: open ga_input_file and read its contents
- output: dictionary (ga_input_dict) that contains the values listed in State Variables
- exception: FileNotFoundError := ga_input_file $\notin$ current working directory

verify_ga_input(ga_input_dict):

- transition: None
- output: None
- exception: ValueError
  - 4 >num_atoms
  - num_filled >num_slots
  - num_swaps >num_geoms
  - t_size >num_filled $\lor$ t_size <2
  - not an integer type (for all except coef_energy and coef_rmsd, which should be floats)
  - missing key/unknown key
  - too many keys (i.e. repeated keys), where num_args $\neq$ NUM_GA_ARGS

6.4.5 Local Functions

None

7 MIS of Molecule Input

[You can reference SRS labels, such as R1. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

The purpose of this module is to provide a utility for reading and verifying input related to the molecule, including its structure and energy calculations. There are two main functions: read_mol_input and verify_mol_input. The first function opens a data file (.txt file) with the following format:

```
qcm = hartree-fock
basis = aug-cc-pvtz
struct_input = C=CC=C
struct_type = smiles
prog = psi4
charge = 0
multip = 1
```

These values are read into a Python dictionary, called mol_input_dict. The order of the inputs does not matter, but Kaplan will throw an error if one of the keys is missing. This dictionary is then passed to the second function, which checks that the values are correct and that all keys have been given. To verify the molecular input, Vetee's Parser object is constructed using the mol_input_dict (geometry module, Section 15). The mol_input module calls the energy module (Section 16) to run a calculation on the input molecule. If this calculation converges, then the manipulation of the dihedral angles are more likely to afford calculations that converge. After this final verification, the Parser object is passed back to the gac module, and eventually gets used by the pmem module (Section 13). From the SRS document, QCM and BS are represented here as qcm and basis respectively. All keys and string values are case insensitive.

7.1 Module

mol_input

7.2 Uses

geometry (Section 15), energy (Section 16)

7.3 Syntax

7.3.1 Exported Constants

NUM_MOL_ARGS := 7

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
read_mol_input	str	dict	FileNotFoundError
verify_mol_input	dict	Parser	ValueError

7.4 Semantics

7.4.1 State Variables

num_args $:= +(x : list | key \in x \wedge value \in x \wedge length(x) = 2 : 1)$

mol_input_dict, which is a dictionary that contains:

- qcm := str $\in$ available methods given prog
- basis := str $\in$ available basis sets given prog and molecule
- struct_input := $\{x : str | x = file \vee x = SMILES \vee x = name \vee x = cid : x\}$
- struct_type := $str \in \{\text{“}smiles\text{”}, \text{“}xyz\text{”}, \text{“}com\text{”}, \text{“}glog\text{”}, \text{“}name\text{”}, \text{“}cid\text{”}\}$
- prog := $str \in \{\text{“}psi4\text{”}\}$
- charge := $\mathbb{Z}$
- multip := $\mathbb{N}$

7.4.2 Environment Variables

mol_input_file: str representing the file that exists in the working directory (optionally includes a prepended path).

7.4.3 Assumptions

As with 6, other modules that use and exchange the state variables found in this module will not raise errors related to the type of input.

7.4.4 Access Routine Semantics

read_mol_input(mol_input_file):

- transition: open mol_input_file and read its contents
- output: dictionary (mol_input_dict) that contains the values listed in State Variables
- exception: FileNotFoundError := mol_input_file $\notin$ current working directory

verify_mol_input(mol_input_dict):

- transition: None
- output: Parser
- exception: ValueError
  - qcm $\notin$ prog
  - basis $\notin$ prog $\vee$ basis unavailable for molecule
  - unable to parse SMILES string, name, cid, or input file
  - struct_type not available
  - not a string type (for all except charge (integer) and multip (natural number))
  - missing key/unknown key
  - too many keys (i.e. repeated keys), where num_args $\neq$ NUM_MOL_ARGS

7.4.5 Local Functions

None

8 MIS of GA Control

This module is responsible for running the GA using the given inputs. The general format of the algorithm is as follows:

1. Read in and verify ga_input_file (6).
2. Read in and verify mol_input_file (7). This step includes a check of the initial geometry, QCM, and BS for convergence (16), and generating a Parser object (15).
3. Generate a Ring object (12).
4. Fill the Ring with Pmem objects according to the num_filled input variable (13).
5. Iterate over the num_mevs input variable, and run a tournament on the Ring according to the t_size variable (10).
6. Return the output as per the output module specifications (14).

8.1 Module

gac

8.2 Uses

ga_input (Section 6), mol_input (Section 7), output (Section 14), ring (Section 12), tournament (Section 10)

8.3 Syntax

8.3.1 Exported Constants

8.3.2 Exported Access Programs

[Not sure if I should list the exceptions raised by imported modules here? For example, reading the input may give an error, but this error is not explicitly raised by the gac module. Do I still have to list it here? —JG]

Name	In	Out	Exceptions
run_kaplan	str, str	None	None

8.4 Semantics

8.4.1 State Variables

[I don't actually need to store the mol_input_dict variable here since its information will be completely contained in the Parser object. Mostly I am leaving this note here as a reminder to update the Parser object with the prog attribute. This update would also mean consolidating the function calls read and verify mol input. Not sure if it is a good idea to have these separated in gac? —JG]

- ga_input_dict : dict (see 6)
- mol_input_dict : dict (see 7)
- parser : Parser (see 15)
- ring : Ring (see 12)
- mev : $\{x : \mathbb{Z} | \text{mum\_mevs} > x \geq 0 : x\}$

8.4.2 Environment Variables

None

8.4.3 Assumptions

This is the main control unit for the program; the user will write their own input files and only need to access this module to complete their task.

8.4.4 Access Routine Semantics

run_kaplan(ga_input_file, mol_input_file):

- transition: The desired conformers are produced and passed to the output module (14).
- output: None
- exception: None

8.4.5 Local Functions

None [Since the run_kaplan function is only used by gac, should I put it here in local functions? Technically the user will have to import it somewhere to use it. —JG]

9 MIS of Fit_G

[You can reference SRS labels, such as R1. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

9.1 Module

fitg

9.2 Uses

energy (Section 16), rmsd (Section 17)

9.3 Syntax

I am using Pmem to indicate usage of the Pmem class. Lowercase p - pmem - will be used to indicate that this is a specific instance of the Pmem class.

9.3.1 Exported Constants

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-
sum_energies	Pmem	float	-
sum_rmsd	Pmem	float	-
fitness	int, float, float, Pmem	float	-
fit_form0	float, float, float, float	float	-

9.4 Semantics

9.4.1 State Variables

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

9.4.2 Environment Variables

None

9.4.3 Assumptions

New fitness functions will be added. Each new fitness functions should be incrementally labelled. The first fitness function is called fit_form0. Subsequent functions will be called fit_form1, fit_form2, etc. Any new fitness functions that are added should account for the possibility of divide by zero errors.

9.4.4 Access Routine Semantics

sum_energies(pmem): $pmem \rightarrow \mathbb{R}$

- transition: [if appropriate —SS]
- output: out := $+(x : \mathbb{R} | x \in$ pmem.energies : $|x|)$
- exception: [if appropriate —SS]

sum_rmsd(pmem): $pmem \rightarrow \mathbb{R}$

- transition: [if appropriate —SS]
- output: out := $+(x : \mathbb{R} | x \in$ pmem.rmsd $\wedge\, x \neq$ None : $|x|)$
- exception: [if appropriate —SS]

fitness(pmem, fit_form, coef_energy, coef_rmsd):

- transition: [if appropriate —SS]
- output: out := ()
- exception: [if appropriate —SS]

fit_form0(sum_energy, sum_rmsd, coef_energy, coef_rmsd):

- transition: [if appropriate —SS]
- output: out := (coef_energy*sum_energy + coef_rmsd*sum_rmsd)
- exception: [if appropriate —SS]

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[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]

9.4.5 Local Functions

[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]

10 MIS of Tournament

The tournament module selects t_size pmems from the ring for comparison. It ranks the pmems in order of increasing fitness, as calculated using the fitg module (9). Then, the two best pmems are chosen as "parents", and the mutations module (11) is used to generate two new "children" based on these parents. The ring module (12) then decides whether the children are added to the ring or not, and if old pmems are deleted to make room for the children.

10.1 Module

tournament

10.2 Uses

ring (Section 12), pmem (Section 13), mutations (Section 11)

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
run_tournament	$\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Z}$, Ring	None	EmptyRingError
select_pmems	$\mathbb{N}$, Ring	list	None
select_parents	list, Ring	list	None

10.4 Semantics

10.4.1 State Variables

- selected_pmems := $\{x : \mathbb{N} | x \geq 0 \wedge \text{ring.pmems}[x] \neq \mathit{None}\}$
- parents := $\{x : \mathbb{N} | x \geq 0 \wedge \text{ring.pmems}[x] \neq \mathit{None}\}$
- parent1 := $[[D_1], [D_2], ..., [D_{n_G}]]$, where each D_i is a list of length num_atoms of integers representing the dihedral angles
- parent2 := $[[D_1], [D_2], ..., [D_{n_G}]]$, where each D_i is a list of length num_atoms of integers representing the dihedral angles
- children := $[[[D_1], [D_2], ..., [D_{n_G}]], [[D_1], [D_2], ..., [D_{n_G}]]]$, where each D_i is a list of length num_atoms of integers representing the dihedral angles

10.4.2 Environment Variables

None

10.4.3 Assumptions

None [May have to think on this more. —JG]

10.4.4 Access Routine Semantics

run_tournament(t_size, num_muts, num_swaps, ring):

- transition: new ring members may be added (changes ring.num_filled), deleted (replace old pmem with a new one)
- output: None
- exception: EmptyRingError when t_size ¿ ring.num_filled

10.4.5 Local Functions

select_pmems(number, ring):

- transition: None
- output: selection, which is a list of length *number* of random ring indices containing a pmem; no pmem can appear twice in the selection
- exception: None

select_parents(selected_pmems, ring):

- transition: None
- output: list of length 2 representing the ring indices of the pmems with the best fitness out of the selected_pmems indicies according to fitness value
- exception: None

11 MIS of Crossover & Mutation

This module is used to generate new solution instances with which to generate new pmems (13) for the ring (12). It has two local functions, mutate and swap. Mutate takes in a list of lists representing dihedral angles for the molecule of interest. Then, a number of these dihedral angles are randomly changed. Swap takes in two of such list of lists and swaps a number of sublists between the two inputs. This module is called during a tournament (10). A wrapper function to call these two functions is called generate_children, which returns the new solution instances to the tournament.

11.1 Module

mutations

11.2 Uses

None

11.3 Syntax

11.3.1 Exported Constants

These two values restrict the dihedral angles that can be chosen.
MIN_VALUE : 0
MAX_VALUE : 360

11.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
generate_children	list(list($\mathbb{Z}$)), list(list($\mathbb{Z}$)), $\mathbb{Z}$, $\mathbb{Z}$	list(list($\mathbb{Z}$)), list(list($\mathbb{Z}$))	None
mutate	list(list($\mathbb{Z}$)), $\mathbb{Z}$	list(list($\mathbb{Z}$))	None
swap	list(list($\mathbb{Z}$)), list(list($\mathbb{Z}$)), $\mathbb{Z}$	list(list($\mathbb{Z}$)), list(list($\mathbb{Z}$))	None

11.4 Semantics

11.4.1 State Variables

- number of chosen swaps
- number of chosen mutations

11.4.2 Environment Variables

None

11.4.3 Assumptions

None [Maybe good to reference SRS assumptions about inputs here? —JG]

11.4.4 Access Routine Semantics

generate_children(parent1, parent2, num_muts, num_swaps):

- transition: None
- output: two list of lists of integers between MIN_VALUE and MAX_VALUE representing dihedrals for two new population members. These will be used to create pmem objects.
- exception: None

11.4.5 Local Functions

mutate(dihedrals, num_muts):

- transition: locally update num_muts with a random number (min 0, max num_muts)
- output: a list of lists of integers between MIN_VALUE and MAX_VALUE representing dihedrals for one mutated population member.
- exception: None

swap(parent1, parent2, num_swaps):

- transition: locally update num_swaps with a random number (min 0, max num_swaps)
- output: two list of lists of integers between MIN_VALUE and MAX_VALUE representing dihedrals for two swapped population members.
- exception: None

12 MIS of Ring

[You can reference SRS labels, such as R1. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

12.1 Module

[Short name for the module —SS]

12.2 Uses

12.3 Syntax

12.3.1 Exported Constants

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

12.4 Semantics

12.4.1 State Variables

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

12.4.2 Environment Variables

[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]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[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

[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 Pmem

[You can reference SRS labels, such as R1. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

13.1 Module

pmem

13.2 Uses

13.3 Syntax

13.3.1 Exported Constants

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

13.4 Semantics

13.4.1 State Variables

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

13.4.2 Environment Variables

[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]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[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

[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 Output

The output module is called by the GA Control module (8) to produce output for the program. There are two main requirements: return the best set of conformer geometries with full geometry specifications and calculate their respective energies. Although it is not listed in the requirements, this module will most likely be upgraded to include some statistical measurements of the results.

14.1 Module

output

14.2 Uses

geometry (Section 15), ring (Section 12), pmem (Section 13)

14.3 Syntax

14.3.1 Exported Constants

OUTPUT_FORMAT = "xyz"

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
run_output	ring	None	None

14.4 Semantics

14.4.1 State Variables

- total_fit : the total sum of all fitness values for pmems in the Ring (that aren't None), float ≥ 0
- best_pmem : the ring index for the pmem in the ring with the highest fitness value, $\mathbb{Z} \geq 0$
- best_fit : the value of the best fitness as found in the ring, float ≥ 0

14.4.2 Environment Variables

The output files with extension OUTPUT_FORMAT will be written to the current working directory.

14.4.3 Assumptions

None

14.4.4 Access Routine Semantics

run_output(ring):

- transition: generate an output files for the best conformer geometries.
- output: returns the average fitness value in the ring, the best fitness value in the ring, the energies of the best geometries, and the final ring object.
- exception: raise an error if the user doesn't have write permissions in the current working directory.

14.4.5 Local Functions

None

15 MIS of Geometry

[You can reference SRS labels, such as R1. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

15.1 Module

geometry

15.2 Uses

15.3 Syntax

15.3.1 Exported Constants

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

15.4 Semantics

15.4.1 State Variables

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

15.4.2 Environment Variables

[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]

15.4.3 Assumptions

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

15.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[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]

15.4.5 Local Functions

[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]

16 MIS of Energies

[You can reference SRS labels, such as R1. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

16.1 Module

energy

16.2 Uses

16.3 Syntax

16.3.1 Exported Constants

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

16.4 Semantics

16.4.1 State Variables

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

16.4.2 Environment Variables

[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]

16.4.3 Assumptions

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

16.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[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]

16.4.5 Local Functions

[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]

17 MIS of RMSD

[You can reference SRS labels, such as R1. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

17.1 Module

[Short name for the module —SS]

17.2 Uses

17.3 Syntax

17.3.1 Exported Constants

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

17.4 Semantics

17.4.1 State Variables

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

17.4.2 Environment Variables

[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]

17.4.3 Assumptions

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

17.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[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]

17.4.5 Local Functions

[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]

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.

18 Appendix

[Extra information if required —SS]