Module Interface Specification for CVT Simulator

Team #17, Baja Dynamics
Grace McKenna
Travis Wing
Cameron Dunn
Kai Arseneau

January 16, 2025

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at [give url —SS]

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

Contents

3 Introduction

The following document details the Module Interface Specifications for the CVT Simulator-program which is designed for optimizing McMaster Baja vehicles. This document specifies how each module interacts with one another throughout the program.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/gr812b/CVT-Simulator.

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 CVT Simulator.

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 (-∞, ∞)
positive real	$\mathbf{R}_+$	any real number ($\mathbf{R}$) in (0, ∞)

The specification of CVT Simulator 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, CVT Simulator 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	Engine Simulator Module External Forces Module CVT Simulation Module Input Module ODE Solver Module Main Module Playback Module Visualizer Module Constants Module State Module Backend Controller Module
Software Decision Module	GUI Module File Output Module Communication Module

Table 1: Module Hierarchy

6 Engine Simulator Module

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

6.1 Module

Engine Module

6.2 Uses

- Constants Module (14)

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
getTorque	angularVeloctiy (ℝ)	torque (ℝ)	-
calcuAngularAccel	angularVeloctiy (ℝ), loadTorque (ℝ)	angularAcceleration (ℝ)	-

6.4 Semantics

6.4.1 State Variables

- Torque curve $\mathbb{R} \to \mathbb{R}$
- Inertia $\mathbb{R}$

6.4.2 Environment Variables

None

6.4.3 Assumptions

- Torque Curve is initialized from the constants module
- Inertia is positive

6.4.4 Access Routine Semantics

getTorque(angularVeloctiy):

- output: torque:= torqueCurve(angularVeloctiy)

calcAngularAccel(angularVeloctiy, loadTorque):

- output: angularAcceleration:= (loadTorque - getTorque(angularVeloctiy))/inertia

6.4.5 Local Functions

None

7 External Forces Module

7.1 Module

Load Simulator

7.2 Uses

- Constants Module (14)

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
calcInclineForce	-	inclineForce $\mathbb{F}$	-
calcDragForce	velocity $\mathbb{F}$	dragForce $\mathbb{F}$	-
calcLoadTorque	velocity $\mathbb{F}$	loadTorque $\mathbb{F}$	-
calcGearboxLoad	velocity $\mathbb{F}$	gearboxLoad $\mathbb{F}$	-

7.4 Semantics

7.4.1 State Variables

None

7.4.2 Environment Variables

None

7.4.3 Assumptions

Constants are initialized from the constants module

7.4.4 Access Routine Semantics

calcInclineForce():

- output: inclineForce:= carMass*gravity*sin(inclineAngle)

calcDragForce():

- output: dragForce:= 0.5*airDensity*frontalArea*dragCoefficient*velocity^2

calcLoadTorque():

- output: loadTorque:= dragForce + inclineForce

calcGearboxLoad():

- output: gearboxLoad:= (loadTorque*wheelRadius)/gearboxRatio

7.4.5 Local Functions

None

8 MIS of CVT Simulation Module

[Use labels for cross-referencing —SS]

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

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

8.1 Module

[Short name for the module —SS]

8.2 Uses

- Constants Module (14)

8.3 Syntax

8.3.1 Exported Constants

8.3.2 Exported Access Programs

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

8.4 Semantics

8.4.1 State Variables

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

8.4.2 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]

8.4.3 Assumptions

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

8.4.4 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]

8.4.5 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]

9 MIS of Input Module

[Use labels for cross-referencing —SS]

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

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

9.1 Module

[Short name for the module —SS]

9.2 Uses

None.

9.3 Syntax

9.3.1 Exported Constants

9.3.2 Exported Access Programs

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

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

[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]

9.4.3 Assumptions

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

9.4.4 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]

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 ODE Solver Module

10.1 Module

ODE Solver

10.2 Uses

- Constants Module (14)
- CVT Simulation Module (8)
- External Forces Module (7)
- Engine Simulator Module (6)
- State Module (15)

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
solveIvp	list of enginerAngularAcceleration $\mathbb{R}$ and carAccleration $\mathbb{R}$ and carVelocity $\mathbb{S}$, timeSpan (tuple of $\mathbb{Z}$ and $\mathbb{Z}$), array of $\mathbb{S}$, array of $\mathbb{F}$	list of States (State Module)	-

10.4 Semantics

10.4.1 State Variables

None

10.4.2 Environment Variables

None

10.4.3 Assumptions

None

10.4.4 Access Routine Semantics

solveIVP():

- output: states:= solve_ivp(ode, initial_state, time_span, args)

10.4.5 Local Functions

None

11 MIS of Main Module

11.1 Module

Main

11.2 Uses

- Communication Module (19)
- Visualizer Module (13)

11.3 Syntax

11.3.1 Exported Constants

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
main	-	-	-

11.4 Semantics

11.4.1 State Variables

None

11.4.2 Environment Variables

None

11.4.3 Assumptions

The GUI module is assumed to be running in the background and is used to display the results of the simulation.

11.4.4 Access Routine Semantics

main():

- transition: Connects the backend controller module to the visualizer module.

11.4.5 Local Functions

None

12 MIS of Playback Module

[Use labels for cross-referencing —SS]

[You can reference SRS labels, such as R??. —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

None.

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 Visualizer Module

[Use labels for cross-referencing —SS]

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

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

13.1 Module

[Short name for the module —SS]

13.2 Uses

- GUI Module (17)

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 Constants Module

14.1 Module

Constants

14.2 Uses

None.

14.3 Syntax

14.3.1 Exported Constants

- `ENGINE_INERTIA`: A positive real value ($\mathbb{R}_+$) representing the inertia of the current car's engine (in $\text{kg} \cdot \text{m}^2$) used for calculations involving car specifications.
- `GEARBOX_RATIO`: A positive real value ($\mathbb{R}_+$) representing the current car's gearbox ratio (unitless) used for calculations involving car specifications.
- `FRONTAL_AREA`: A positive real value ($\mathbb{R}_+$) representing the current car's frontal area (in m^2) used for calculations involving car specifications.
- `DRAG_COEFFICIENT`: A positive real value ($\mathbb{R}_+$) representing the current car's drag coefficient (unitless) used for calculations involving car specifications.
- `CAR_WEIGHT`: A positive real value ($\mathbb{R}_+$) representing the current car's weight (in lbs) used for calculations involving car specifications.
- `CAR_MASS`: A positive real value ($\mathbb{R}_+$) representing the current car's weight converted to kilograms (in kg) used for calculations involving car specifications.
- `WHEEL_RADIUS`: A positive real value ($\mathbb{R}_+$) representing the current car's wheel radius (in m) used for calculations involving car specifications.
- `AIR_DENSITY`: A positive real value ($\mathbb{R}_+$), set at `1.225` (in kg/m^3).
- `GRAVITY`: A positive real value ($\mathbb{R}_+$), set at `9.80665` (in m/s^2).
- `engineSpecs` A list of dictionaries representing various engine rpm's and corresponding torque values (in ft*lbs):=["rpm": 2400, "torque": 18.5, "rpm": 2600, "torque": 18.1, "rpm": 2800, "torque": 17.4, "rpm": 3000, "torque": 16.6, "rpm": 3200, "torque": 15.4, "rpm": 3400, "torque": 14.5, "rpm": 3600, "torque": 13.5]
- `engineData`: A list of dictionary values for angular velocity(in rad/s), torque(in N*m), and power(torque*angular velocity) converting the above `engineSpecs` into SI units.

- `angular_velocities`: A list of angular velocity values (in rad/s) extracted from `engineData`.
- `torques`: A list of torque values (in N*m) extracted from `engineData`.
- `powers`: A list of power values (in watts) calculated from `engineData`.
- `torque_curve`: A cubic interpolation function that maps `angular_velocities` to `torques`, created using the `interp1d` method with extrapolation for values outside the range.

14.3.2 Exported Access Programs

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

14.4 Semantics

14.4.1 State Variables

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

14.4.2 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]

14.4.3 Assumptions

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

14.4.4 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]

14.4.5 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]

15 MIS of State Module

[Use labels for cross-referencing —SS]

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

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

15.1 Module

[Short name for the module —SS]

15.2 Uses

None.

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 Backend Controller Module

[Use labels for cross-referencing —SS]

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

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

16.1 Module

[Short name for the module —SS]

16.2 Uses

- Input Module (9)
- ODE Solver Module (10)

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 GUI Module

17.1 Module

gui

17.2 Uses

None.

17.3 Syntax

17.3.1 Exported Constants

None.

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
gui	None	None	-

17.4 Semantics

17.4.1 State Variables

- Button states (Boolean for clicked state)
- Input Fields

17.4.2 Environment Variables

- Keyboard ($\mathbf{Z}_+$ for keycodes describing the key pressed)
- Mouse (Boolean for click state and $\mathbf{Z}_+$ for cursor position)
- Screen ($\mathbf{Z}_+$ for width and height in pixels)

17.4.3 Assumptions

None.

17.4.4 Access Routine Semantics

gui():

- transition: Provides methods from Unity to build and deploy a GUI to the Visualizer Module 13

17.4.5 Local Functions

None.

18 MIS of File Output Module

18.1 Module

output

18.2 Uses

None.

18.3 Syntax

18.3.1 Exported Constants

None.

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
write	outputPath (String)	-	-

18.4 Semantics

18.4.1 State Variables

- states: $\mathbb{S}^n$, where each entry represents the state of the car at a given time.

18.4.2 Environment Variables

None.

18.4.3 Assumptions

The file path given can be written to.

18.4.4 Access Routine Semantics

write(outputPath):

- output: Writes the states to a file at the given path.
- exception: [if appropriate —SS]

18.4.5 Local Functions

None.

19 MIS of Communication Module

19.1 Module

communication

19.2 Uses

- Backend Controller Module (16)

19.3 Syntax

19.3.1 Exported Constants

None.

19.3.2 Exported Access Programs

Name	In	Out	Exceptions
frontToBack	primWeight ($\mathbf{R}_+$), primRampGeo ($\mathbb{R} \to \mathbb{R}$), primSpringRate ($\mathbf{R}_+$), primSpringPre ($\mathbf{R}_+$), secHelixGeo ($\mathbb{R} \to \mathbb{R}$), secSpringRate ($\mathbf{R}_+$), secSpringPre ($\mathbf{R}_+$), vehicleWeight ($\mathbf{R}_+$), driverWeight ($\mathbf{R}_+$), traction ($\mathbf{R}_+$), inclineAngle ($\mathbf{R}_+$)	primWeight ($\mathbf{R}_+$), primRampGeo ($\mathbb{R} \to \mathbb{R}$), primSpringRate ($\mathbf{R}_+$), primSpringPre ($\mathbf{R}_+$), secHelixGeo ($\mathbb{R} \to \mathbb{R}$), secSpringRate ($\mathbf{R}_+$), secSpringPre ($\mathbf{R}_+$), vehicleWeight ($\mathbf{R}_+$), driverWeight ($\mathbf{R}_+$), traction ($\mathbf{R}_+$), inclineAngle ($\mathbf{R}_+$)	-
backToFront	-	states ($\mathbb{S}^n$)	-

19.4 Semantics

19.4.1 State Variables

- mainPath: a String representing the path to the main file.
- outputPath: a String representing the path to the file to be read.

19.4.2 Environment Variables

- pythonPath: a String representing the path to the python environment.

19.4.3 Assumptions

All files are in the correct location matching the given paths.

19.4.4 Access Routine Semantics

frontToBack(primWeight, primRampGeo, primSpringRate, primSpringPre, secHelixGeo, secSpringRate, secSpringPre, vehicleWeight, driverWeight, traction, inclineAngle):

- transition: Sends the given parameters to the backend controller.
- exception: [if appropriate —SS]

backToFront():

- transition: Reads the states from the output file.
- exception: [if appropriate —SS]

19.4.5 Local Functions

None.

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.

20 Appendix

[Extra information if required —SS]

Appendix — Reflection

[Not required for CAS 741 projects —SS]

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

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

2. What pain points did you experience during this deliverable, and how did you resolve them?

3. Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from?

4. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), it any, needed to be changed, and why?

5. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)

6. 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)