Module Interface Specification for CVT Simulator

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

January 17, 2025

1 Revision History

Date	Version	Notes
January 16	1.0	Initial Version

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/gr812b/CVT-Simulator/blob/develop/docs/SRS/SRS.pdf

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, ∞)
input	$\mathbb{I}$	a set of values $\{\mathbf{R}_+, \mathbb{R} \rightarrow \mathbb{R}, \mathbf{R}_+, \mathbf{R}_+, \mathbb{R} \rightarrow \mathbb{R}, \mathbf{R}_+, \mathbf{R}_+, \mathbf{R}_+, \mathbf{R}_+, \mathbf{R}_+, \mathbf{R}_+\}$ that represent the input of the program
state	$\mathbb{S}$	a set of values {} representing the state of the simulation
dataPoint	$\mathbb{D}$	Tuple of Time: $\mathbb{R}$, Position: $\mathbb{R}$

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 Initialize 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

6.1 Module

Engine Module

6.2 Uses

- Constants Module (13)

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
getTorque	angularVeloctiy ($\mathbb{R}$)	torque ($\mathbb{R}$)	-
calcuAngularAccel	angularVeloctiy ($\mathbb{R}$), loadTorque ($\mathbb{R}$)	angularAcceleration ($\mathbb{R}$)	-

6.4 Semantics

6.4.1 State Variables

- Torque curve $\mathbb{R} \rightarrow \mathbb{R}$
- 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 (13)

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

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

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

7.5 Module

Primary Module

7.6 Uses

- Constants Module (13)

7.7 Syntax

7.7.1 Exported Constants

7.7.2 Exported Access Programs

Name	In	Out	Exceptions
springForce	shiftDistance ($\mathbb{R}$)	force ($\mathbb{R}$)	-
rampForce	shiftDistance ($\mathbb{R}$), angularVelocity ($\mathbb{R}$)	force ($\mathbb{R}$)	

7.8 Semantics

7.8.1 State Variables

1. flyweight radius ($\mathbb{R}$) (Initial) denoted as r_{fly}
2. Height of flyweights given by shift distance ($\mathbb{R} \rightarrow \mathbb{R}$) denoted as $f_{\text{prim\_height}}(d)$
3. Angle of flyweights given by shift distance ($\mathbb{R} \rightarrow \mathbb{R}$) denoted as $f_{\text{prim\_angle}}(d)$
4. spring coefficient ($\mathbb{R}$) denoted as k_{prim}
5. Initial compression of primary spring ($\mathbb{R}$) denoted as d_{prim}
6. flyweight mass ($\mathbb{R}$) denoted as m_{fly}

7.8.2 Environment Variables

None.

7.8.3 Assumptions

7.8.4 Access Routine Semantics

springForce():

- Output: force $:=$ springForce($k_{\text{prim}}, d_{\text{prim}} + d_{\text{shift}}$)

rampForce():

- output: force $:=$ centForce($m_{\text{fly}}, r_{\text{fly}}$, angularVelocity) $\times$ tan($f_{\text{prim\_angle}}$(shiftDistance))

7.8.5 Local Functions

- springForce: springForce(k, x) $\coloneqq$ k $\times$ x
- centrifugal force: centForce(m, r, w) $\coloneqq$ m $\times$ r $\times$ w^2

7.9 Module

Secondary Module

7.10 Uses

- Constants Module (13)

7.11 Syntax

7.11.1 Exported Constants

7.11.2 Exported Access Programs

Name	In	Out	Exceptions
torsSpringForce	shiftDistance ($\mathbb{R}$)	force ($\mathbb{R}$)	-
compSpringForce	shiftDistance ($\mathbb{R}$)	force ($\mathbb{R}$)	-
rampForce	torque ($\mathbb{R}$), ratio ($\mathbb{R}$), shiftDistance ($\mathbb{R}$)	force ($\mathbb{R}$)	

7.12 Semantics

7.12.1 State Variables

1. springCoefficientTor ($\mathbb{R}$) denoted as k_{tors}
2. springCoefficientComp ($\mathbb{R}$) denoted as k_{comp}
3. Initial torsional rotation of secondary spring ($\mathbb{R}$) denoted as θ_{sec}
4. Initial compression of secondary spring ($\mathbb{R}$) denoted as d_{sec}
5. helix radius ($\mathbb{R}$) denoted as r_{helix}
6. Torsional distance given by distance shifted ($\mathbb{R} \to \mathbb{R}$) denoted as $f_{\text{sec}}(d)$
7. Ramp angle given by distance shifted ($\mathbb{R} \to \mathbb{R}$) denoted as $g_{\text{sec}}(d)$

7.12.2 Environment Variables

None.

7.12.3 Assumptions

7.12.4 Access Routine Semantics

torsSpringForce():

- Output: force $\coloneqq$ torsForce$(k_{\text{tors}}, \theta_{\text{sec}} + f_{\text{sec}}(\text{shiftDistance}))/(2 \cdot r_{\text{helix}} \cdot \tan(g_{\text{sec}}(\text{shiftDistance})))$

compSpringForce():

- output: force $\coloneqq$ springForce$(k_{\text{comp}}, d_{\text{comp}} + \text{shiftDistance})$

helixForce():

- output: force $\coloneqq$ helixForce$(\text{torque}, \text{ratio}, r_{\text{helix}}, g_{\text{sec}}(\text{shiftDistance}))$

7.12.5 Local Functions

- comp spring force: springForce(k, x) $\coloneqq$ k $\cdot$ x
- tors spring force: torsForce$(k, \theta, r) \coloneqq \frac{k \cdot \theta}{r}$
- helix force: helixForce$(T, R, r, \theta) \coloneqq \frac{T \cdot R}{2r \tan(\theta)}$

8 MIS of Initialize Module

8.1 Module

initializer

8.2 Uses

None.

8.3 Syntax

8.3.1 Exported Constants

None.

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
parse	receivedInput ($\mathbb{I}$)	parsedInput ($\mathbb{I}$)	-
initialize	input ($\mathbb{I}$)	state ($\mathbb{S}$)	-

8.4 Semantics

8.4.1 State Variables

None.

8.4.2 Environment Variables

None.

8.4.3 Assumptions

None.

8.4.4 Access Routine Semantics

parse(receivedInput):

- transition: parses the received input into the appropriate format.
- exception: [if appropriate —SS]

initialize(input):

- output: converts input into the initial state of the simulation.
- exception: [if appropriate —SS]

8.4.5 Local Functions

None.

9 MIS of ODE Solver Module

9.1 Module

ODE Solver

9.2 Uses

- Constants Module (13)
- CVT Simulation Module (??)
- External Forces Module (7)
- Engine Simulator Module (6)
- State Module (14)

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
simulate	initialState ($\mathbb{S}$)	result ($\mathbb{S}^n$)	-

9.4 Semantics

9.4.1 State Variables

- time: a tuple of ($\mathbf{R}_+$, $\mathbf{R}_+$) representing the start and end time of the simulation
- step: a value of $\mathbf{R}_+$ representing the time step of the simulation

9.4.2 Environment Variables

None

9.4.3 Assumptions

None

9.4.4 Access Routine Semantics

simulate(initialState):

- output: simulates the ODEs of the simulation for the given initial state, time and step size and returns the result.

9.4.5 Local Functions

None

10 MIS of Main Module

10.1 Module

Main

10.2 Uses

- Communication Module (18)
- Visualizer Module (12)

10.3 Syntax

10.3.1 Exported Constants

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
main	-	-	-

10.4 Semantics

10.4.1 State Variables

None

10.4.2 Environment Variables

None

10.4.3 Assumptions

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

10.4.4 Access Routine Semantics

main():

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

10.4.5 Local Functions

None

11 MIS of Playback Module

11.1 Module

Playback

11.2 Uses

None.

11.3 Syntax

11.3.1 Exported Constants

carSpinTransform: Transform

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
StartPlayback	-	-	-
RestartPlayback	-	-	-
PausePlayback	-	-	-
PlaybackCoroutine	-	-	-

11.4 Semantics

11.4.1 State Variables

- isPlaying: $\mathbb{B}$
- currentIndex: $\mathbb{Z}$
- startTime: $\mathbb{R}$
- carTransform: Transform

11.4.2 Environment Variables

- Start Button: Button
- Restart Button: Button
- Pause Button: Button

11.4.3 Assumptions

Assume that there is data to playback.

11.4.4 Access Routine Semantics

StartPlayback():

- transition: isPlaying:= True, currentIndex:= 0, startTime:= time.time()

PausePlayback():

- transition: isPlaying:= False

RestartPlayback():

- transition: isPlaying:= False, currentIndex:= 0, startTime:= time.time(), carTransform:= back to start position

PlaybackCoroutine():

- transition: carTransform updates to new positions, carSpinTransform updates to new rotations based on position

11.4.5 Local Functions

None

12 MIS of Visualizer Module

12.1 Module

Visualizer

12.2 Uses

- GUI Module (16)
- Playback Module (11)

12.3 Syntax

12.3.1 Exported Constants

None

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
LoadCsvData	-	-	-

12.4 Semantics

12.4.1 State Variables

- dataPoints: list of $\mathbb{D}$

12.4.2 Environment Variables

None

12.4.3 Assumptions

Assume that the simulation results are stored in a csv file.

12.4.4 Access Routine Semantics

LoadCsvData():

- transition: dataPoints:= load data from csv file
- output: dataPoints

None

12.4.5 Local Functions

None

13 MIS of Constants Module

13.1 Module

Constants

13.2 Uses

None.

13.3 Syntax

13.3.1 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.

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
constants	-	-	-

13.4 Semantics

13.4.1 State Variables

None.

13.4.2 Environment Variables

None.

13.4.3 Assumptions

None.

13.4.4 Access Routine Semantics

None.

13.4.5 Local Functions

None.

14 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]

14.1 Module

[Short name for the module —SS]

14.2 Uses

None.

14.3 Syntax

14.3.1 Exported Constants

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 Backend Controller Module

15.1 Module

Backend Controller

15.2 Uses

- Initialize Module (8)
- ODE Solver Module (9)
- File Output Module (17)

15.3 Syntax

15.3.1 Exported Constants

None

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
main	-	-	-

15.4 Semantics

15.4.1 State Variables

None

15.4.2 Environment Variables

None

15.4.3 Assumptions

Assume that the other modules are functioning correctly.

15.4.4 Access Routine Semantics

main():

- transition: Connects the different parts of the backend together

15.4.5 Local Functions

None

16 MIS of GUI Module

16.1 Module

`gui`

16.2 Uses

None.

16.3 Syntax

16.3.1 Exported Constants

None.

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
gui	None	None	-

16.4 Semantics

16.4.1 State Variables

- Button states (Boolean for clicked state)
- Input Fields ($\mathbb{I}$)

16.4.2 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)

16.4.3 Assumptions

None.

16.4.4 Access Routine Semantics

`gui`():

- transition: Provides methods from Unity to build and deploy a GUI to the Visualizer Module 12

16.4.5 Local Functions

None.

17 MIS of File Output Module

17.1 Module

output

17.2 Uses

None.

17.3 Syntax

17.3.1 Exported Constants

None.

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
write	outputPath (String)	-	-

17.4 Semantics

17.4.1 State Variables

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

17.4.2 Environment Variables

None.

17.4.3 Assumptions

The file path given can be written to.

17.4.4 Access Routine Semantics

write(outputPath):

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

17.4.5 Local Functions

None.

18 MIS of Communication Module

18.1 Module

communication

18.2 Uses

- Backend Controller Module (15)

18.3 Syntax

18.3.1 Exported Constants

None.

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
frontToBack	input ($\mathbb{I}$)	ouput ($\mathbb{I}$)	-
backToFront	-	states ($\mathbb{S}^n$)	-

18.4 Semantics

18.4.1 State Variables

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

18.4.2 Environment Variables

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

18.4.3 Assumptions

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

18.4.4 Access Routine Semantics

frontToBack(input):

- 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]

18.4.5 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.

19 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?
   During this deliverable the team was successfully able to document our design choices for the CVT Simulator. Creating this document helped identify gaps in our initial planning and lead our team to have a clearer idea and breakdown of our program going into Rev 0. By organizing our design choices we now have a strong foundation and structured documentation to build on, setting us up for success in Rev 0 and future revisions.

2. What pain points did you experience during this deliverable, and how did you resolve them?
   The main pain point our group faced was the discovered need to expand large modules, that had not been fully scoped out yet. While writing these large modules this lead to the realization that having one module for these large parts did not make sense. This pain point was resolved by realizing it was necessary to create additional modules and introduce helper functions within these modules to help simplify the expansion and organization of the module.

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?
   The design of the UI and what would be required by our interfaces was guided by client feedback. Based on discussions with the McMaster Baja team we knew how our interface should look and function. The other design decisions that did not stem from client input, the team relied on our own experiences and expertise.

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?

During the creation of this document our team did not need to change any other parts of any existing document.

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)
   The two primary limitation of our teams' solution are the modelling accuracy and expertise in advanced mathematics. By increasing our computational power and having more precise measurements of time within the simulation would allow for more complex simulations with a higher precision.

6. Give a brief overview of other design solutions you considered. What are the benefits and trade offs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)
   Our team considered using alternative platforms such as Unreal Engine and React for the application. Although Unreal Engine would have likely worked well for the project needs this software was dismissed due it's cost. React was an additional option discussed, however the current communication protocol would need to be revised away from using CSV files. Unity and Python were chosen as the most feasible and cost-effective solution, aligning with the projects constraints and needs.