Module Interface Specification for MTOBridge

Team 15, Alpha Software Solutions
Badawy, Adham
Yazdinia, Pedram
Jandric, David
Vakili, Farzad
Vezina, Victor
Chiu, Darren

April 4, 2023

1 Revision History

Date	Developer(s)	Notes
January 15, 2023	David and Darren	Initial Draft
January 17, 2023	Adham, Farzad, Pedram, and Victor	Conversion to LaTeX
January 18, 2023	David and Victor	Formatting
April 4, 2023	Darren	Rev1 Changes

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/agentvv/MTOBridge/blob/main/docs/SRS/SRS.pdf.

2.1 Abbreviations and Acronyms

symbol	description
MIS	Module Interface Specification
SRS	Software Requirements Specification

Contents

3 Introduction

The following document details the Module Interface Specifications for MTOBridge,

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

4 Notation

The structure of the MIS for modules comes from **?**, with the addition that template modules have been adapted from **?**. The mathematical notation comes from Chapter 3 of **?**. For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

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

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 number	$\mathbb{R}$	any number in (-∞, ∞)

The specification of MTOBridge 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, MTOBridge 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 is taken directly from the Module Guide document for this project.

- **Hardware Hiding**
    - Data Storage Hiding
        - Input Data Storage Hiding
            - **Truck Platoon Configuration Data Storage Hiding**
            - **Bridge Configuration Data Storage Hiding**
        - Calculation Data Storage Hiding

      - Calculation Variable Data Storage Hiding
        - **Solver Choice Data Storage Hiding**
        - **Concerned Section Specification Data Storage Hiding**
        - **Critical Section Specification Data Storage Hiding**
      - **Output Report Data Storage Hiding**
  - File System Hiding
    - **Save To File System Hiding**
    - **Load From File System Hiding**

- **Software Decision Hiding**
  - **Calculation Call Hiding**
  - **MATLAB Code Hiding**
  - Data Format Hiding
    - Configuration Data Format Hiding
      - **Truck Platoon Configuration Data Format Hiding**
      - **Bridge Configuration Data Format Hiding**
      - **Solver Configuration Data Format Hiding**
    - **Report Data Format Hiding**

- **Behavior Hiding**
  - Visualization Hiding
    - Configuration Visualization Hiding
      - **Truck Platoon Configuration Visualization Hiding**
      - **Bridge Configuration Visualization Hiding**
    - Calculation Visualization Hiding
      - **Calculation Call Visualization Hiding**
      - **Solver Configuration Visualization Hiding**
    - **Output Report Visualization Hiding**
    - File System Visualization Hiding
      - Save To File System Visualization Hiding
        - **Truck Platoon Configuration Save To File System Visualization Hiding**
        - **Bridge Configuration Save To File System Visualization Hiding**
        - **Solver Configuration Save To File System Visualization Hiding**
        - **Output Report Save To File System Visualization Hiding**

- Load From File System Visualization Hiding
  - **Truck Platoon Configuration Load From File System Visualization Hiding**
  - **Bridge Configuration Load From File System Visualization Hiding**
  - **Output Report Load From File System Visualization Hiding**

6 MIS of Save to File System

6.1 Module

Saver

6.2 Uses

PlatoonConfiguration, BridgeConfiguration, SolverConfiguration, Report

6.3 Syntax

6.3.1 Exported Constants

None

6.4 Exported Types

None

6.4.1 Exported Access Programs

Name	In	Out	Exceptions
savePlatoonConfiguration	PlatoonT, string	-	-
saveBridgeConfiguration	BridgeT, string	-	-
saveSolverConfiguration	SolverT, string	-	-
saveReport	ReportT, string	-	-

6.5 Semantics

6.5.1 State Variables

None

6.5.2 Environment Variables

None

6.5.3 Assumptions

None

6.5.4 Access Routine Semantics

savePlatoonConfiguration(platoon, filepath)

- output: open outputFile at filepath for write and write each field of platoon to outputFile delineated by ", " in the following order: platoon.axleLoad as a string of space separated numbers, platoon.axleSpacing as a string of space separated numbers, number of trucks, headway Close outputFile.

- exception: none

saveBridgeConfiguration(bridge, filepath)

- output: open outputFile at filepath for write and write each field of bridge to outputFile delineated by ", " in the following order: bridge.numberSpans, bridge.spanLength as a string of space separated numbers, bridge.concernedSection, bridge.discretizationLength Close outputFile.

- exception: none

saveSolverConfiguration(solver, filepath)

- output: open outputFile at filepath for write and write each field of platoon to outputFile delineated by ", " in the following order: solver.forceType, solver.solverType Close outputFile.

- exception: none

saveReport(report, filepath)

- output: open outputFile at path for write. Write a header for the truck platoon configuration in the form "[Platoon]", then write all fields of truck platoon in the form of "field.name = field.value". Do the same for bridge and solver configurations, with their respective headers and fields. Then, write a header for the analysis in the form "[Results]". If the solver was configured for critical sections, write "critical section = value". Proceed to write out the pairs of first axle position and force, separated by commas and each pair separated by new lines. Close the output file.

- exception: none

6.5.5 Local Functions

None

7 MIS of Load From File System

7.1 Module

Loader

7.2 Uses

PlatoonConfiguration, BridgeConfiguration, SolverConfiguration

7.3 Syntax

7.3.1 Exported Constants

None

7.4 Exported Types

None

7.4.1 Exported Access Programs

Name	In	Out	Exceptions
loadPlatoonConfiguration	string	PlatoonT	-
loadBridgeConfiguration	string	BridgeT	-
loadReport	string	{ PlatoonT, BridgeT, SolverT }	-

7.5 Semantics

7.5.1 State Variables

None

7.5.2 Environment Variables

None

7.5.3 Assumptions

None

7.5.4 Access Routine Semantics

loadPlatoonConfiguration(filepath)

- output: open outputFile at filepath for read and read each field: Parse space separated $\mathbb{R}$ until comma as axleLoad, parse next set of space separated $\mathbb{R}$ until the next comma as axleSpacing, parse next $\mathbb{N}$ as numberOfTrucks, parse last $\mathbb{R}$ as headway. Close outputFile.
  out := PlatoonT constructed from the parsed values above.

- exception: none

loadBridgeConfiguration(filepath)

- output: open outputFile at filepath for read and read each field: Parse first number as numberOfSpans, parse space separated $\mathbb{R}$ until comma as spanLength, parse next $\mathbb{R}$ as concernedSection, parse last $\mathbb{R}$ as discretization length. Close outputFile.
  out := BridgeT constructed from the parsed values above.

- exception: none

loadReport(filepath)

- output: open outputFile at filepath for read. Process lines until the line "[Platoon]" is found. Parse the space-separated numbers on the line containing "axleLoad" as the PlatoonT axleLoad. Parse for axleSpacing in the same manner. Parse for headway and number of trucks in the same manner. Parse for BridgeT in the same manner beginning at "[Bridge]." Parse for SolverT in the same manner beginning at "[Solver]." Close outputFile.
  out := { PlatoonT, BridgeT, SolverT } constructed from the parsed values above.

- exception: none

7.5.5 Local Functions

None

8 MIS of Calculation Call

8.1 Module

CalculationCaller

8.2 Uses

PlatoonConfiguration, BridgeConfiguration, SolverConfiguration, Report

8.3 Syntax

8.3.1 Exported Constants

None

8.4 Exported Types

CalculationInputT = {
truckConfig: TruckT,
bridgeConfig: BridgeT,
solverConfig: SolverT
}
CalculationOutputT = {
allForces: sequence of sequence of $\mathbb{R}$,
firstAxlePosition: sequence of $\mathbb{R}$,
forceConcernedSection: sequence of $\mathbb{R}$,
forceCriticalSection: sequence of $\mathbb{R}$,
maxForce: $\mathbb{R}$,
firstAxlePositionMaxForce: $\mathbb{R}$,
sections: sequence of $\mathbb{R}$,
criticalSection: $\mathbb{R}$,
forceEnvelope: sequence of $\mathbb{R}$,
firstAxlePositionForceEnvelope: sequence of $\mathbb{R}$
}

8.4.1 Exported Access Programs

Name	In	Out	Exceptions
runCalculation	CalculationInputT	CalculationOutputT	-

8.5 Semantics

8.5.1 State Variables

None

8.5.2 Environment Variables

engine: An instance of the MATLAB engine which exports functions that run the relevant MATLAB code.

8.5.3 Assumptions

None

8.5.4 Access Routine Semantics

runCalculation(in):

- output: $out :=$
  $(in.solverConfig.solverType = CONCERNED \Rightarrow$
  $concernedSection(in) \mid$
  $in.solverConfig.solverType = CRITICAL \Rightarrow$
  $criticalSection(in)) \wedge$
  $report.updateReport(ReportT\{in, out\})$
- exception: none

8.5.5 Local Functions

concernedSection: $CalculationInputT \rightarrow CalculationOutputT$:

- output: $out :=$
  $(in.bridgeConfig.numberSpans = 1 \Rightarrow$
  $concernedSectionOneSpan(in) \mid$
  $in.bridgeConfig.numberSpans = 2 \Rightarrow$
  $concernedSectionTwoSpan(in) \mid$
  $in.bridgeConfig.numberSpans = 3 \Rightarrow$
  $concernedSectionThreeSpan(in))$
- exception: none

concernedSectionOneSpan: $CalculationInputT \rightarrow CalculationOutputT$:

- output: $out :=$
  $(in.solverConfig.forceType = POSITIVE\_MOMENT \Rightarrow$
  $concernedSectionOneSpanMoment(in) \mid$

$in.solverConfig.forceType = SHEAR \Rightarrow$
$concernedSectionOneSpanShear(in))$

- exception: none

concernedSectionOneSpanMoment: $CalculationInputT \rightarrow CalculationOutputT$:

- output: $out := engine.Concerned\_section\_one\_span\_moment(in)$
- exception: none

concernedSectionOneSpanShear: $CalculationInputT \rightarrow CalculationOutputT$

- output: $out := engine.Concerned\_section\_one\_span\_shear(in)$
- exception: none

concernedSectionTwoSpan: $CalculationInputT \rightarrow CalculationOutputT$

- output: $out :=$
  $(in.solverConfig.forceType = POSITIVE\_MOMENT \Rightarrow$
  $concernedSectionTwoSpanPositiveMoment(in) \mid$
  $in.solverConfig.forceType = NEGATIVE\_MOMENT \Rightarrow$
  $concernedSectionTwoSpanNegativeMoment(in) \mid$
  $in.solverConfig.forceType = SHEAR \Rightarrow$
  $concernedSectionTwoSpanShear(in))$
- exception: none

concernedSectionTwoSpanPositiveMoment: $CalculationInputT \rightarrow CalculationOutputT$

- output: $out := engine.Concerned\_section\_two\_span\_moment(in)$
- exception: none

concernedSectionTwoSpanNegativeMoment: $CalculationInputT \rightarrow CalculationOutputT$

- output: $out := engine.Concerned\_section\_two\_span\_moment(in)$
- exception: none

concernedSectionTwoSpanShear: $CalculationInputT \rightarrow CalculationOutputT$

- output: $out := engine.Concerned\_section\_two\_span\_shear(in)$
- exception: none

concernedSectionThreeSpan: $CalculationInputT \rightarrow CalculationOutputT$

- output: $out :=$
  $(in.solverConfig.forceType = POSITIVE\_MOMENT \Rightarrow$
  $concernedSectionThreeSpanPositiveMoment(in) \mid$
  $in.solverConfig.forceType = NEGATIVE\_MOMENT \Rightarrow$
  $concernedSectionThreeSpanNegativeMoment(in) \mid$
  $in.solverConfig.forceType = SHEAR \Rightarrow$
  $concernedSectionThreeSpanShear(in))$

- exception: none

concernedSectionThreeSpanPositiveMoment: $CalculationInputT \rightarrow CalculationOutputT$

- output: $out := engine.Concerned\_section\_three\_span\_moment(in)$

- exception: none

concernedSectionThreeSpanNegativeMoment: $CalculationInputT \rightarrow CalculationOutputT$

- output: $out := engine.Concerned\_section\_three\_span\_moment(in)$

- exception: none

concernedSectionThreeSpanShear: $CalculationInputT \rightarrow CalculationOutputT$

- output: $out := engine.Concerned\_section\_three\_span\_shear(in)$

- exception: none

criticalSection: $CalculationInputT \rightarrow CalculationOutputT$

- output: $out :=$
  $(in.bridgeConfig.numberSpans = 1 \Rightarrow$
  $criticalSectionOneSpan(in) \mid$
  $in.bridgeConfig.numberSpans = 2 \Rightarrow$
  $criticalSectionTwoSpan(in) \mid$
  $in.bridgeConfig.numberSpans = 3 \Rightarrow$
  $criticalSectionThreeSpan(in))$

- exception: none

criticalSectionOneSpan: $CalculationInputT \rightarrow CalculationOutputT$

- output: $out :=$
  $(in.solverConfig.forceType = POSITIVE\_MOMENT \Rightarrow$
  $criticalSectionOneSpanMoment(in) \mid$
  $in.solverConfig.forceType = SHEAR \Rightarrow$
  $criticalSectionOneSpanShear(in))$

- exception: none

criticalSectionOneSpanMoment: $CalculationInputT \rightarrow CalculationOutputT$

- output: $out := engine.Critical\_section\_one\_span\_moment(in)$
- exception: none

criticalSectionOneSpanShear: $CalculationInputT \rightarrow CalculationOutputT$

- output: $out := Critical\_section\_one\_span\_shear(in)$
- exception: none

criticalSectionTwoSpan: $CalculationInputT \rightarrow CalculationOutputT$

- output: $out :=$
  $(in.solverConfig.forceType = POSITIVE\_MOMENT) \Rightarrow$
  $criticalSectionTwoSpanPositiveMoment(in) \mid$
  $in.solverConfig.forceType = NEGATIVE\_MOMENT) \Rightarrow$
  $criticalSectionTwoSpanNegativeMoment(in) \mid$
  $in.solverConfig.forceType = SHEAR \Rightarrow$
  $criticalSectionTwoSpanShear(in))$
- exception: none

criticalSectionTwoSpanPositiveMoment: $CalculationInputT \rightarrow CalculationOutputT$

- output: $out := engine.Critical\_section\_two\_span\_positive\_moment(in)$
- exception: none

criticalSectionTwoSpanNegativeMoment: $CalculationInputT \rightarrow CalculationOutputT$

- output: $out := engine.Critical\_section\_two\_span\_negative\_moment(in)$
- exception: none

criticalSectionTwoSpanShear: $CalculationInputT \rightarrow CalculationOutputT$

- output: $out := engine.Critical\_section\_two\_span\_shear(in)$
- exception: none

criticalSectionThreeSpan: $CalculationInputT \rightarrow CalculationOutputT$

- output: $out :=$
  $(in.solverConfig.forceType = POSITIVE\_MOMENT \Rightarrow$
  $criticalSectionThreeSpanPositiveMoment(in) \mid$
  $in.solverConfig.forceType = NEGATIVE\_MOMENT) \Rightarrow$
  $criticalSectionThreeSpanNegativeMoment(in) \mid$
  $in.solverConfig.forceType = SHEAR \Rightarrow$
  $criticalSectionThreeSpanShear(in))$

- exception: none

criticalSectionThreeSpanPositiveMoment: $CalculationInputT \rightarrow CalculationOutputT$

- output: $out := engine.Critical\_section\_three\_span\_positive\_moment(in)$
- exception: none

criticalSectionThreeSpanNegativeMoment: $CalculationInputT \rightarrow CalculationOutputT$

- output: $out := engine.Critical\_section\_three\_span\_negative\_moment(in)$
- exception: none

criticalSectionThreeSpanShea: $CalculationInputT \rightarrow CalculationOutputT$

- output: $out := engine.Critical\_section\_three\_span\_shear(in)$
- exception: none

9 MIS of Truck Platoon Configuration Data Format

9.1 Module

PlatoonConfiguration

9.2 Uses

None

9.3 Syntax

9.3.1 Exported Constants

None

9.4 Exported Types

PlatoonT = {
axleLoad: sequence of $\mathbb{R}$,
axleSpacing: sequence of $\mathbb{R}$,
numberOfTrucks: $\mathbb{N}$,
headway: $\mathbb{R}$
}

9.4.1 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
new PlatoonConfiguration	-	PlatoonConfiguration	-
getConfiguration	-	PlatoonT	-
updateAxleLoad	seq of $\mathbb{R}$	-	-
updateAxleSpacing	seq of $\mathbb{R}$	-	-
updateNumberOfTrucks	$\mathbb{N}$	-	-
updateHeadway	$\mathbb{R}$	-	-

9.5 Semantics

9.5.1 State Variables

axleLoad: sequence of $\mathbb{R}$,
axleSpacing: sequence of $\mathbb{R}$,

numberOfTrucks: $\mathbb{N}$,
headway: $\mathbb{R}$

9.5.2 Environment Variables

None

9.5.3 Assumptions

None

9.5.4 Access Routine Semantics

new PlatoonConfiguration():

- transition:
  axleLoad := $\emptyset$
  axleSpacing := $\emptyset$
  numberOfTrucks := 0
  headway := 0.0
- output: out := self
- exception: none

PlatoonT getConfiguration():

- output: out := PlatoonT { axleLoad, axleSpacing, numberOfTrucks, headway }
- exception: none

updateAxleLoad(newAxleLoad):

- transition: self.axleLoad := newAxleLoad
- exception: none

updateAxleSpacing(newAxleSpacing):

- transition: self.axleSpacing := newAxleSpacing
- exception: none

updateNumberOfTrucks(newNumberOfTrucks)

- transition: self.numberOfTrucks := newNumberOfTrucks
- exception: none

updateHeadway(newHeadway)

- transition: self.headway := newHeadway
- exception: none

9.5.5 Local Functions

None

10 MIS of Bridge Configuration Data Format

10.1 Module

BridgeConfiguration

10.2 Uses

None

10.3 Syntax

10.3.1 Exported Constants

None

10.4 Exported Types

BridgeT = {
numberSpans: {1, 2, 3},
spanLength: seq of $\mathbb{R}$,
concernedSection: $\mathbb{R}$,
discretizationLength: $\mathbb{R}$,
}

10.4.1 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
new BridgeConfiguration	-	BridgeConfiguration	-
getConfiguration	-	BridgeT	-
updateNumberOfSpans	string	-	invalidConfigurationValue
updateSpanLength	string	-	invalidConfigurationValue
updateConcernedSection	string	-	invalidConfigurationValue
updateDiscretizationLength	string	-	invalidConfigurationValue

10.5 Semantics

10.5.1 State Variables

numberOfSpans: $\mathbb{N}$
spanLength: seq of $\mathbb{R}$

concernedSection: $\mathbb{R}$
discretizationLength: $\mathbb{R}$

10.5.2 Environment Variables

None

10.5.3 Assumptions

None

10.5.4 Access Routine Semantics

new BridgeConfiguration():

- transition:
  numberOfSpans := 0
  spanLength := empty sequence
  concernedSection := 0.0
  discretizationLength := 0.0
- output: out := self
- exception: none

BridgeT getConfiguration():

- output: out := BridgeT { numberOfSpans, spanLength, concernedSection, discretizationLength }
- exception: none

updateNumberOfSpans(newNumberOfSpans):

- transition: Parse newNumberOfSpans as $\mathbb{N}$. Set self.numberOfSpans to the parsed value of newNumberOfSpans.
- exception: $exc := newNumberOfSpans \notin \mathbb{N} \Rightarrow invalidConfigurationValue$

updateSpanLength(newSpanLength):

- transition: Parse newSpanLength as a space-separated sequence of $\mathbb{R}$. Set self.axleLoad to the parsed value of newAxleLoad.
- exception: $exc := \exists(n|n \in newSpanLength \wedge (n \notin \mathbb{N} \wedge n \neq \text{' '})) \Rightarrow invalidConfigurationValue$

updateConcernedSection(newConcernedSection)

- transition: Parse newConcernedSection as $\mathbb{R}$. Set self.concernedSection to the parsed value of newConcernedSection.
- exception: $exc := newConcernedSection \notin \mathbb{R} \Rightarrow invalidConfigurationValue$

updateDiscretizationLength(newDiscretizationLength)

- transition: Parse newDiscretizationLength as $\mathbb{R}$. Set self.discretizationLength to the parsed value of newDiscretizationLength.
- exception: $exc := newDiscretizationLength \notin \mathbb{R} \Rightarrow invalidConfigurationValue$

10.5.5 Local Functions

None

11 MIS of Solver Configuration Data Format

11.1 Module

SolverConfiguration

11.2 Uses

None

11.3 Syntax

11.3.1 Exported Constants

None

11.4 Exported Types

SolverT = {
 forceType: {POSITIVE_MOMENT, NEGATIVE_MOMENT, SHEAR},
 solverType: {CONCERNED, CRITICAL}
}

11.4.1 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
new SolverConfiguration	-	SolverConfiguration	-
getConfiguration	-	SolverT	-
updateForceType	string	-	invalidConfigurationValue
updateSolverType	string	-	invalidConfigurationValue

11.5 Semantics

11.5.1 State Variables

forceType: SolverT.forceType
solverType: SolverT.solverType

11.5.2 Environment Variables

None

11.5.3 Assumptions

None

11.5.4 Access Routine Semantics

new SolverConfiguration():

- transition:
  forceType := POSITIVE_MOMENT
  solverType := CONCERNED
- output: out := self
- exception: none

getConfiguration():

- transition: none
- output: out := SolverT { self.forceType, self.solverType}
- exception:none

updateForceType(newForceType):

- transition:
  (newForceType = "Positive Moment" ⇒ forceType := POSITIVE_MOMENT |
  newForceType = "Negative Moment" ⇒ forceType := NEGATIVE_MOMENT |
  newForceType = "Shear" ⇒ forceType := SHEAR)
- exception: $exc := newForceType \notin \{$"Positive Moment", "Negative Moment", "Shear"$\} \Rightarrow invalidConfigurationValue$

updateSolverType(newSolverType):

- transition:
  (newSolverType = "Concerned Section" ⇒ solverType := CONCERNED |
  newSolverType = "Negative Moment" ⇒ solverType := CRITICAL)
- exception: $exc := newSolverType \notin \{$"Concerned Section", "Negative Moment"$\} \Rightarrow invalidConfigurationValue$

11.5.5 Local Functions

None

12 MIS of Report Data Format

12.1 Module

Report

12.2 Uses

None

12.3 Syntax

12.3.1 Exported Constants

None

12.4 Exported Types

ReportT = {
 input: CalculationInputT
 results: CalculationOutputT
}

12.4.1 Exported Access Programs

Name	In	Out	Exceptions
new Report	-	Report	-
getReport	-	ReportT	invalidReport
updateReport	ReportT	-	-

12.5 Semantics

12.5.1 State Variables

report: ReportT

12.5.2 Environment Variables

None

12.5.3 Assumptions

None

12.5.4 Access Routine Semantics

new Report():

- transition: report := NULL
- output: out := self
- exception: none

getReport():

- output: out := report
- exception: report = NULL ⇒ invalidReport

updateReport(ReportT newReport):

- transition: report := newReport
- exception: none

12.5.5 Local Functions

None

13 MIS of Truck Platoon Configuration Save To File System Visualization

13.1 Module

TruckSaver

13.2 Uses

Saver, PlatoonConfiguration

13.3 Syntax

13.3.1 Exported Constants

None

13.4 Exported Types

None

13.4.1 Exported Access Programs

Name	In	Out	Exceptions
new TruckSaver	-	TruckSaver	-
buttonPressed	-	-	-

13.5 Semantics

13.5.1 State Variables

None

13.5.2 Environment Variables

None

13.5.3 Assumptions

None

13.5.4 Access Routine Semantics

new TruckSaver()

- output: out := TruckSaver
- exception: none

buttonPressed()

- output: When the button is clicked, open a windows file dialog to select where to save the file. After the user selects a valid location and clicks save, the method shall call PlatoonConfiguration().getConfiguration() to get the current truck configuration. Then, the method shall call savePlatoonConfiguration with the parsed configuration and the filepath the user specified.
- exception: none

13.5.5 Local Functions

None

14 MIS of Bridge Configuration Save To File System Visualization

14.1 Module

BridgeSaver

14.2 Uses

Saver, BridgeConfiguration

14.3 Syntax

14.3.1 Exported Constants

None

14.4 Exported Types

None

14.4.1 Exported Access Programs

Name	In	Out	Exceptions
new BridgeSaver	-	BridgeSaver	-
buttonPressed	-	-	-

14.5 Semantics

14.5.1 State Variables

None

14.5.2 Environment Variables

None

14.5.3 Assumptions

None

14.5.4 Access Routine Semantics

new BridgeSaver()

- output: out := BridgeSaver
- exception: none

buttonPressed()

- output: When the button is clicked, open a windows file dialog to select where to save the file. After the user selects a valid location and clicks save, the method shall call BridgeConfiguration().getConfiguration() to get the current bridge configuration. Then, the method shall call saveBridgeConfiguration with the parsed configuration and the filepath the user specified.
- exception: none

14.5.5 Local Functions

None

15 MIS of Solver Configuration Save To File System Visualization

15.1 Module

SolverSaver

15.2 Uses

Saver, SolverConfiguration, Report

15.3 Syntax

15.3.1 Exported Constants

None

15.4 Exported Types

None

15.4.1 Exported Access Programs

Name	In	Out	Exceptions
new SolverSaver	-	SolverSaver	-
buttonPressed	-	-	-

15.5 Semantics

15.5.1 State Variables

None

15.5.2 Environment Variables

None

15.5.3 Assumptions

None

15.5.4 Access Routine Semantics

new SolverSaver()

- output: out := SolverSaver
- exception: none

buttonPressed()

- output: When the button is clicked, open a windows file dialog to select where to save the file. After the user selects a valid location and clicks save, the method shall call Report.getReport() to get the current results, plot them, and then save the graph to the filepath the user specified.
- exception: none

15.5.5 Local Functions

None

16 MIS of Output Report Save To File System Visualization

16.1 Module

ReportSaver

16.2 Uses

Saver, Report

16.3 Syntax

16.3.1 Exported Constants

None

16.4 Exported Types

None

16.4.1 Exported Access Programs

Name	In	Out	Exceptions
new ReportSaver	-	ReportSaver	-
buttonPressed	-	-	-

16.5 Semantics

16.5.1 State Variables

None

16.5.2 Environment Variables

None

16.5.3 Assumptions

None

16.5.4 Access Routine Semantics

new ReportSaver()

- output: out := ReportSaver
- exception: none

buttonPressed()

- output: When the button is clicked, open a windows file dialog to select where to save the file. After the user selects a valid location and clicks save, the method shall get the most recently generated output report with Report().getReport(). Then, the method shall call saveReport with the parsed report and the filepath the user specified.
- exception: none

16.5.5 Local Functions

None

17 MIS of Truck Platoon Configuration Load From File System Visualization

17.1 Module

TruckLoader

17.2 Uses

Loader, PlatoonConfiguration

17.3 Syntax

17.3.1 Exported Constants

None

17.4 Exported Types

None

17.4.1 Exported Access Programs

Name	In	Out	Exceptions
new TruckLoader	-	TruckLoader	-
buttonPressed	-	-	-

17.5 Semantics

17.5.1 State Variables

None

17.5.2 Environment Variables

None

17.5.3 Assumptions

None

17.5.4 Access Routine Semantics

new ReportSaver()

- output: out := TruckLoader
- exception: none

buttonPressed()

- output: When the button is clicked, open a windows file dialog to select the file containing the platoon configuration. After the user selects a file and clicks load, the method shall call loadPlatoonConfiguration() with the file specified by the user. Then, the method shall set the values of TruckConfiguration() with the respective values.
- exception: none

17.5.5 Local Functions

None

18 MIS of Bridge Configuration Load From File System Visualization

18.1 Module

BridgeLoader

18.2 Uses

Loader, BridgeConfiguration

18.3 Syntax

18.3.1 Exported Constants

None

18.4 Exported Types

None

18.4.1 Exported Access Programs

Name	In	Out	Exceptions
new BridgeLoader	-	BridgeLoader	-
buttonPressed	-	-	-

18.5 Semantics

18.5.1 State Variables

None

18.5.2 Environment Variables

None

18.5.3 Assumptions

None

18.5.4 Access Routine Semantics

new BridgeLoader()

- output: out := BridgeLoader
- exception: none

buttonPressed()

- output: When the button is clicked, open a windows file dialog to select the file containing the bridge configuration. After the user selects a file and clicks load, the method shall call loadBridgeConfiguration() with the file specified by the user. Then, the method shall set the values of BridgeConfiguration() with the respective values.
- exception: none

18.5.5 Local Functions

None

19 MIS of Output Report Load From File System Visualization

19.1 Module

ReportLoader

19.2 Uses

Loader, Report

19.3 Syntax

19.3.1 Exported Constants

None

19.4 Exported Types

None

19.4.1 Exported Access Programs

Name	In	Out	Exceptions
new ReportLoader	-	ReportLoader	-
buttonPressed	-	-	-

19.5 Semantics

19.5.1 State Variables

None

19.5.2 Environment Variables

None

19.5.3 Assumptions

None

19.5.4 Access Routine Semantics

new ReportLoader()

- output: out := ReportLoader
- exception: none

buttonPressed()

- output: When the button is clicked, open a windows file dialog to select the file containing the report. After the user selects a file and clicks load, the method shall call loadReport() with the file specified by the user. Then, the method shall set the values of PlatoonConfiguration, BridgeConfiguration, and SolverConfiguration with the respective values.
- exception: none

19.5.5 Local Functions

None

20 MIS of Truck Platoon Configuration Visualization

20.1 Module

PlatoonVisualizer

20.2 Uses

PlatoonConfiguration

20.3 Syntax

20.3.1 Exported Constants

None

20.4 Exported Types

None

20.4.1 Exported Access Programs

Name	In	Out	Exceptions
new PlatoonVisualizer	-	PlatoonVisualizer	-
numAxlesChanged	$\mathbb{N}$	-	-
platoonConfigured	-	-	-
visualize	-	-	-

20.5 Semantics

20.5.1 State Variables

numAxles: $\mathbb{N}$
axleLoad: sequence of $\mathbb{R}$
axleSpacing: sequence of $\mathbb{R}$
numberOfTrucks: $\mathbb{N}$
headway: $\mathbb{R}$

20.5.2 State Invariants

$numAxles \geq 3 \wedge numAxles \leq 17$
$|axleLoad| \leq numberOfTrucks * numAxles$
$|axleSpacing| \leq numberOfTrucks * (numAxles - 1)$

$\forall(x : \mathbb{R} | x \in axleLoad \wedge x \geq 0.1 \wedge x \leq 1000)$
$\forall(x : \mathbb{R} | x \in axleSpacing \wedge x \geq 1.2 \wedge x \leq 20)$
$numberOfTrucks > 0$
$headway \geq 5.0$

20.5.3 Environment Variables

None

20.5.4 Assumptions

None

20.5.5 Access Routine Semantics

new PlatoonVisualizer()

- transition:
  numAxles := 3
  axleLoad := ∅
  axleSpacing := ∅
  numberOfTrucks := 1
  headway := 5.0
- output: out := self
- exception: none

numAxlesChanged(i)

- transition: numAxles := i. Add or remove input fields until there are i axle load fields and $i - 1$ axle spacing fields. Call platoonConfigured.
- exception: none

platoonConfigured()

- transition: If
  $|axleLoad| = numberOfTrucks * numAxles \wedge$
  $|axleSpacing| = numberOfTrucks * (numAxles - 1)$
  call
  TruckConfiguration.updateAxleLoad(axleLoad)
  TruckConfiguration.updateAxleSpacing(axleSpacing)
  TruckConfiguration.updateNumberOfTrucks(numberOfTrucks)
  TruckConfiguration.updateHeadway(headway)

- exception: none

visualize()

- output: Call TruckConfiguration.getConfig(), and do the following with the result:
    - Draw truck based on truck.axleSpacing.
    - Then add an offset of truck.headway.
    - Do this truck.numberOfTrucks times.

  Draw the fields for axleLoad, axleSpacing, numberOfTrucks, headway using the respective state variables as their values.
- exception: none

20.5.6 Local Functions

None

21 MIS of Bridge Configuration Visualization

21.1 Module

BridgeVisualizer

21.2 Uses

BridgeConfiguration

21.3 Syntax

21.3.1 Exported Constants

None

21.4 Exported Types

None

21.4.1 Exported Access Programs

Name	In	Out	Exceptions
new BridgeVisualizer	-	BridgeVisualizer	-
numberOfSpansDetermined	seq of $\mathbb{R}$	-	-
bridgeConfigEdited	-	-	-
visualize	-	-	-

21.5 Semantics

21.5.1 State Variables

numberOfSpans: $\mathbb{N}$
spanLength: $\mathbb{R}$
concernedSection: $\mathbb{R}$
discretizationLength: $\mathbb{R}$

21.5.2 State Invariants

$numberOfSpans \geq 1 \land numberOfSpans \leq 3$
$|spanLength| \leq numberOfSpans$
$\forall(x : \mathbb{R} | x \in spanLength \land x \geq 0.0)$

$concernedSection \geq 0.0 \wedge concernedSection \leq spanLength$
$discretizationLength \geq 0.0$

21.5.3 Environment Variables

None

21.5.4 Assumptions

None

21.5.5 Access Routine Semantics

new BridgeVisualizer()

- transition:
  numberOfSpans := 1
  spanLength := 0.0
  concernedSection := 0.0
  discretizationLength := 0.0
- output: out := self
- exception: none

numberOfSpansDetermined(numberOfSpans)

- transition: Whenever the user updates the numberOfSpans field, set self.numberOfSpans := numberOfSpans. Add or remove span length input fields until there are $numberOfSpans$ fields. Call bridgeConfigEdited().
- exception: none

bridgeConfigEdited()

- transition: Whenever the user updates an input field, if $|spanLength| = numberOfSpans$, convert numberOfSpans, spanLength, concernedSection, and discretizationLength into strings and call

  BridgeConfiguration.updateNumberOfSpans(numberOfSpans)

  BridgeConfiguration.updateSpanLength(spanLength)

  BridgeConfiguration.updateConcernedSection(concernedSection)

  BridgeConfiguration.updateDiscretizationLength(discretizationLength)
- exception: none

visualize()

- output: Call BridgeConfiguration.getConfig(), and do the following with the result:
  - Draw bridge using bridge.numberOfSpans and bridge.spanLength.
  - If bridge.discretizationLength is defined, then draw the separate bridge sections of size bridge.discretizationLength.
  - If bridge.sectionOfConcern is defined, then show a vertical line at the position of bridge.sectionOfConcern.

  Draw the input fields for numberOfSpans, spanLength, concernedSection, discretizationLength using the respective state variables as their values.
- exception: none

21.5.6 Local Functions

None

22 MIS of Solver Configuration Visualization

22.1 Module

SolverVisualizer

22.2 Uses

SolverConfiguration

22.3 Syntax

22.3.1 Exported Constants

None

22.4 Exported Types

None

22.4.1 Exported Access Programs

Name	In	Out	Exceptions
new SolverVisualizer	-	SolverVisualizer	-
setForceType	-	-	invalidConfigurationValue
setSolverType	-	-	invalidConfigurationValue
visualize	-	-	-

22.5 Semantics

22.5.1 State Variables

forceType: string
solverType: string

22.5.2 Environment Variables

None

22.5.3 Assumptions

None

22.5.4 Access Routine Semantics

new SolverVisualizer()

- transition:
  forceType := ""
  solverType := ""
- output: out := self
- exception: none

setForceType()

- transition: Whenever the user updates the forceType field, set the forceType string to the value of the field, and call SolverConfiguration.updateForceType(forceType).
- exception: SolverConfiguration.updateForceType(forceType) could not parse forceType ⇒ invalidConfigurationValue

setSolverType()

- transition: Whenever the user updates the solverType field, set the solverType string to the value of the field, and call SolverConfiguration.updateSolverType(solverType).
- exception: SolverConfiguration.updateSolverType(solverType) could not parse solverType ⇒ invalidConfigurationValue

visualize()

- output: Draw the input fields for forceType, solverType using the respective state variables as their values.
- exception: none

22.5.5 Local Functions

None

23 MIS of Calculation Call Visualization

23.1 Module

CalculationCallVisualizer

23.2 Uses

CalculationCall

23.3 Syntax

23.3.1 Exported Constants

None

23.4 Exported Types

None

23.4.1 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
new CalculationCallVisualizer	-	CalculationCallVisualizer	-
buttonPressed	-	-	invalidConfiguration

23.5 Semantics

23.5.1 State Variables

None

23.5.2 Environment Variables

None

23.5.3 Assumptions

None

23.5.4 Access Routine Semantics

new CalculationCallVisualizer()

- transition: none
- output: out := self
- exception: none

buttonPressed()

- output: When the button to calculate is pressed, call CalculationCaller().runCalculation((CalculationInputT) { TruckConfiguration.getConfig(), BridgeConfiguration.getConfig(), SolverConfiguration.getConfig() }).
- exception: if TruckConfiguration(), BridgeConfiguration() are invalid configurations ⇒ invalidConfiguration

23.5.5 Local Functions

None

24 MIS of Output Report Visualization

24.1 Module

ReportVisualizer

24.2 Uses

Report

24.3 Syntax

24.3.1 Exported Constants

None

24.4 Exported Types

None

24.4.1 Exported Access Programs

Name	In	Out	Exceptions
new ReportVisualizer	-	ReportVisualizer	-
visualize	-	-	invalidReport

24.5 Semantics

24.5.1 State Variables

None

24.5.2 Environment Variables

None

24.5.3 Assumptions

None

24.5.4 Access Routine Semantics

new ReportVisualizer()

- output: out := self
- exception: none

visualize()

- output: Call Report.getReport(). To visualize the ReportT, show each field of the input alongside its output. Then, to visualize the results, show a depiction of the truck platoon and bridge above a graph. The graph should have an x-axis of the first axle positions, and the y-axis should be the force at each first axle position. For critical section analysis, there should be a second graph showing the moment envelope with bridge section on the x-axis and moment on the y-axis. Also, there should be a vertical line where the critical/concerned section is on the bridge.
- exception: There is no report ⇒ invalidReport

24.5.5 Local Functions

None