Module Interface Specification for SFWRENG 4G06 Capstone Design Project

Team #18, InfiniView-AI
Anhao Jiao
Kehao Huang
Qianlin Chen
Qi Shu
Xunzhou Ye

January 17, 2024

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 [Fill in your project name and description —SS]

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at .... [provide the url for your repo —SS]

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 SFWRENG 4G06 Capstone Design Project.

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 (-∞, ∞)

The specification of SFWRENG 4G06 Capstone Design Project 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, SFWRENG 4G06 Capstone Design Project 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	
Behaviour-Hiding	Input Parameters
	Output Format
	Output Verification
	Temperature ODEs
	Energy Equations
	Control Module
	Specification Parameters Module
Software Decision	Sequence Data Structure
	ODE Solver
	Plotting

Table 1: Module Hierarchy

6 MIS of RTC Control Module

6.1 Module

RTCControl

6.2 Uses

Web APIs
STUN Server Module

6.3 Syntax

6.3.1 Exported Constants

N/A

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
createPeerConnection	JSON	RTCPeerConnection	-
closeRemoteConnection	RTCPeerConnection	-	-
negotiate	RTCPeerConnection	-	-

6.4 Semantics

6.4.1 State Variables

N/A

6.4.2 Environment Variables

STUN_SERVER_ADDRESS: string — represents the address of the STUN server
SFU_BROADCAST_API: string — represents the API endpoint for SFU broadcast API
SFU_CONSUME_API: string — represents the API endpoint for SFU consume API

6.4.3 Assumptions

SFU server and STUN servers are running in normal conditions.

6.4.4 Access Routine Semantics

createPeerConnectionWith(config: JSON):

- transition: N/A

- output: pc := RTCPeerConnection — initializes a new RTCPeerConnection based on the given configuration.
- exception: N/A

closeRemoteConneciton(pc: RTCPeerConnection):

- transition: pc.signalingState := closed — closes peer connection and send a signal to the connected peer connection.
- output: N/A
- exception: N/A

negotiate(pc: RTCPeerConnection):

- transition:

  pc.localDescription := RTCSessionDescriptionInit

  pc.remoteDescription := RTCSessionDescriptionInit

  sets the local description of the peer connection to its generated SDP, and set the remote description of the peer connection to its received SDP from SFU_BROADCAST_API.
- output: N/A
- exception: N/A

getRemoteStream(pc: RTCPeerConnection):

- transition: pc.event := getRemoteEvent(pc).streams
- output: N/A
- exception: N/A

6.4.5 Local Functions

getRemoteEvent(pc: RTCPeerConnection):

- transition: N/A
- output: pc.event := RTCTrackEvent
- exception: N/A

7 MIS of Annotation Configuration Module

7.1 Module

AnnotationConfig

7.2 Uses

React
RTC Control Module

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
setIsSkeletonEnabled	Boolean	-	-
setIsCOMEnabled	Boolean	-	-
getIsSkeletonEnabled	-	Boolean	-
getIsCOMEnabled	-	Boolean	-

7.4 Semantics

7.4.1 State Variables

isSkeletonEnabled: Boolean
isCOMEnabled: Boolean

7.4.2 Environment Variables

N/A

7.4.3 Assumptions

N/A

7.4.4 Access Routine Semantics

setIsSkeletonEnabled(isEnabled: Boolean):

- transition: isSkeletonEnabled := isEnabled
- output: N/A
- exception: N/A

setIsCOMEnabled(isEnabled: Boolean):

- transition: isCOMEnabled := isEnabled
- output: N/A
- exception: N/A

getIsSkeletonEnabled():

- input: N/A
- transition: N/A
- output: isSkeletonEnabled
- exception: N/A

getIsCOMEnabled():

- input: N/A
- transition: N/A
- output: isCOMEnabled
- exception: N/A

7.4.5 Local Functions

N/A

8 MIS of App Module

8.1 Module

App

8.2 Uses

RTC Control Module
Media Control Module
Instructor View Module
Practitioner View Module
Annotation Configuration Module
User Authentication Module

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
App	-	React.component	-

8.4 Semantics

8.4.1 State Variables

None

8.4.2 Environment Variables

N/A

8.4.3 Assumptions

N/A

8.4.4 Access Routine Semantics

App():

- transition: App:= React.component() Start React App and render it on the user's device
- output: N/A
- exception: N/A

8.4.5 Local Functions

N/A

9 MIS of User Authentication Module

9.1 Module

Auth

9.2 Uses

Instructor View Module
Practitioner View Module

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
Auth	-	React.component	-

9.4 Semantics

9.4.1 State Variables

isUserInstructor: Boolean

9.4.2 Environment Variables

None

9.4.3 Assumptions

None

9.4.4 Access Routine Semantics

Auth():

- transition: Render the authentication page on the user's device, if the user clicks on the Instructor button, then jumps to the instructor view page, if the user clicks on the practitioner button, jumps to the practitioner view page.
- output: N/A
- exception: N/A

9.4.5 Local Functions

isUserInstructor → Instructor view else Practitioner view
setIsUserInstructor(isEnabled: Boolean):

- transition: isUserInstructor := isEnabled
- output: N/A
- exception: N/A

10 MIS of Video Transform Module

10.1 Module

VideoTransformTrack

10.2 Uses

HPE, CM

10.3 Syntax

10.3.1 Exported Constants

kind = "video"

10.3.2 Exported Access Programs

Routine name	In	Out	Exceptions
__init__	track, transform	-	-
recv	-	VideoFrame	-

10.4 Semantics

10.4.1 State Variables

track: VideoStreamTrack
transform: str

10.4.2 Environment Variables

None

10.4.3 Assumptions

__init__ is called before any other access program

10.4.4 Access Routine Semantics

__init__(track, transform):

- transition: initiated by track and transform, self.track = track, self.transform = transform
- output: out := self
- exception: None

recv(self):

- transition: Processes a video frame (frame) received from a track. Depending on the value of self.transform, it applies one of the following transformations: "HPE": Converts the frame by applying the HPE module annotation. "CM": Converts the frame by applying the CM module annotation. If self.transform is set to any other value, the frame is returned without any transformation.
- output: Returns a new VideoFrame object (new_frame) that has undergone the specified transformation, preserving the original frame's timing information (timestamps and time base).
- exception: None

10.4.5 Local Functions

None

11 MIS of SFU Server Module

11.1 Module

SfuServer

11.2 Uses

VideoTransformTrack

11.3 Syntax

11.3.1 Exported Constants

None

11.3.2 Exported Access Programs

Routine name	In	Out	Exceptions
consumer	request	-	-
broadcast	request	-	-

11.4 Semantics

11.4.1 State Variables

None

11.4.2 Environment Variables

relay: MediaRelay
consumer_track: VideoStreamTrack

11.4.3 Assumptions

None

11.4.4 Access Routine Semantics

consumer(request):

- transition: Processes a WebRTC connection request. Parses the request to extract session description parameters. Creates a new RTCPeerConnection object. Logs the information about the sent track. Adds a VideoTransformTrack to the peer connection, which includes subscribing to a consumer track and applying a specified video transformation. Sets the remote description of the peer connection based on the received session description. Creates and sets a local description for the peer connection by generating an answer to the received offer.
- output: Returns a web response in JSON format. This response contains the SDP data and the type of the local description set on the peer connection. This information is crucial for establishing the WebRTC connection.
- exception: There is no explicit exception handling within the function. If an error occurs during any of the steps (e.g., parsing the request, setting up the peer connection, or creating the response), the function may raise an exception related to that error. However, such exceptions are not explicitly caught or handled within the function itself. Potential errors could arise from invalid request data, failures in peer connection operations, or issues in response generation.

broadcast(request):

- transition: Manages the setup and handling of a WebRTC peer connection for broadcasting. Parses the incoming request to extract the SDP data. Initializes a new RTCPeerConnection. Adds the peer connection to a global set and logs relevant information. Sets up event handlers for different peer connection events:
    1. Connection State Change: Monitors the connection state, logging changes and closing the connection if it fails.
    2. Track Reception: Handles received tracks, particularly video tracks, by setting a global consumer_track for later use, and logs when tracks end.
    3. Processes the received offer by setting it as the remote description of the peer connection.
    4. Creates and sets a local description for the peer connection in response to the offer.
- output: Returns a web response in JSON format, containing the SDP data and the type of the local description set on the peer connection. This is essential for completing the WebRTC connection setup.
- exception: The function does not explicitly handle exceptions. Errors during the processing of the request, peer connection operations, or event handling may result in exceptions. These exceptions are not caught within the function, meaning the caller must handle any arising errors. Potential errors could include issues with the request format, failures in peer connection setup, or problems in event handling.

11.4.5 Local Functions

N/A

12 MIS of Human Pose Estimation Annotation Module

12.1 Module

HPE

12.2 Uses

Numpy, CV2, OS, Sys, Time, Subprocess, Shutil, Socket

12.3 Syntax

12.3.1 Exported Constants

server_address, HPE_address, K, pose, Rt1, R1, t1, P1, Identity, P2

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
get_kpts	Image	List	IOError, ValueError
measureJoint	List, List	Tuple	N/A
matchKpts	List	List	N/A
get3D	List, List	List	N/A

12.4 Semantics

12.4.1 State Variables

N/A

12.4.2 Environment Variables

N/A

12.4.3 Assumptions

External libraries are functioning as expected

12.4.4 Access Routine Semantics

get_kpts(img):

- transition: Saves the input image to a designated path and calls OpenPose to generate keypoints, which are then saved to a JSON file.
- output: Returns a list of keypoints extracted from the input image.
- exception: IOError if image saving or reading fails, ValueError if keypoints processing fails.

measureJoint(kpts1, kpts2):

- transition: Computes the length of the spine in each set of keypoints and returns them ordered by length.

- output: Returns a tuple with the first element being the keypoints set with the longer spine.
- exception: N/A

matchKpts(mirror_img):

- transition: Reflects the keypoints from the mirror image to match the real image.
- output: Returns the adjusted keypoints for the mirrored image.
- exception: N/A

get3D(real_kpts, mirror_kpts):

- transition: Uses the keypoints from the real and mirror images to triangulate 3D points.
- output: Returns the 3D coordinates of the keypoints.
- exception: N/A

12.4.5 Local Functions

N/A: All functions are intended to be accessed by other modules within the system

13 MIS of Center of Mass Annotation Module

13.1 Module

CM

13.2 Uses

numpy: for numerical computations
params.bodySegParams: for body segmentation parameters
params.cameraParams: for camera parameters

13.3 Syntax

13.3.1 Exported Constants

K, pose, P1, P2, R1, t1, R2, t2 - Camera intrinsic and extrinsic parameters, and projection matrices derived from them.
foot_in_air_thresh - Threshold for determining if a foot is in the air.
CoM_foot_thresh - Threshold for determining the supporting foot based on the center of mass.

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
getCoM	points_3D: 3D points array	CoM: Center of Mass point	-
feetStates	CoM: Center of Mass point points_3D: 3D points array	left_foot, right_foot: States of the feet	-

13.4 Semantics

13.4.1 State Variables

N/A: The module does not maintain internal state across invocations.

13.4.2 Environment Variables

N/A: This module does not rely on environment variables for its core functionality.

13.4.3 Assumptions

The module assumes that body segment parameters and camera calibration data provided by the bodySegParams and cameraParams modules are accurate and reliable.

13.4.4 Access Routine Semantics

getCoM(points_3D):

- transition: Calculates the center of mass based on the 3D points of body joints.
- output: Returns the 3D coordinates of the body's center of mass.
- exception: No explicit exception handling within the function.

feetStates(CoM, points_3D):

- transition: Determines the state of each foot (left and right) based on their position relative to the center of mass and the vertical distance from the ground.
- output: Returns a tuple containing two dictionaries, left_foot and right_foot, each indicating whether the respective foot is on the ground and whether it is supporting body weight.
- exception: No explicit exception handling within the function.

13.4.5 Local Functions

N/A: All functions are intended to be accessed by other modules within the system

14 MIS of STUN Server Module

14.1 Module

STUN

14.2 Uses

14.3 Syntax

14.3.1 Exported Constants

STUN_SERVER_ADDRESS

14.3.2 Exported Access Programs

14.4 Semantics

14.4.1 State Variables

N/A: The module does not maintain internal state across invocations.

14.4.2 Environment Variables

N/A: This module does not rely on environment variables for its core functionality.

14.4.3 Assumptions

The module assumes that body segment parameters and camera calibration data provided by the bodySegParams and cameraParams modules are accurate and reliable.

14.4.4 Access Routine Semantics

N/A: No Access Routines are exported from this module

14.4.5 Local Functions

N/A: There is no local function in this module

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.

15 Appendix

[Extra information if required —SS]