Module Interface Specification for Software Engineering

Team 5, GradSight
Willie Pai
Hammad Pathan
Wajdan Faheen
Henushan Balachandran
Zahin Hossain

January 18, 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 https://github.com/PaisWillie/Digital-Composite/blob/main/docs/SRS-Volere/SRS.pdf

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

Contents

3 Introduction

The following document details the Module Interface Specifications for Digital Composite

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/PaisWillie/Digital-Composite/tree/main.

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 Software Engineering.

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 Software Engineering 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, Software Engineering 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 Cloud Module

6.1 Module

6.2 Uses

- Cloud Module
- Data Upload Module
- Output Storage Module
- OCR Processing Module

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
manageCloud	Image	Success/Fail Call	InvalidImageFileServerIsDown

6.4 Semantics

6.4.1 State Variables

None

6.4.2 Environment Variables

None

6.4.3 Assumptions

- Input data is provided in a client-approved format
- AWS is running, cloud services are up

6.4.4 Access Routine Semantics

manageCloud():

- transition: Takes a valid image from input format and processes in cloud.

- output: Returns a success or failure call based on the image processing, s3, and lambda function.
- exception: Throws InvalidImageException for non-compliant input data or ServerIsDown is the cloud is not running.

6.4.5 Local Functions

None

7 MIS of Input Module

7.1 Module

7.2 Uses

- Input module
- UI parsing module

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
convertInput	Image	Metadata Formatted in JSON	InvalidImageFile

7.4 Semantics

7.4.1 State Variables

None

7.4.2 Environment Variables

None

7.4.3 Assumptions

- Input data is provided in a client-approved format
- Metadata keys and structure adhere to predefined standards

7.4.4 Access Routine Semantics

convertInput():

- transition: Validates and transforms the input data into the required format.
- output: Returns the formatted data suitable for S3 upload and Lambda processing.
- exception: Throws InvalidFormatException for non-compliant input data.

7.4.5 Local Functions

None

8 MIS of Data Upload Module

8.1 Module

8.2 Uses

- Data Upload Module
- User Interface Parsing Module

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
uploadData	Formatted Image	SuccessResponse	S3Exception, AccessDeniedException, TimeoutException

8.4 Semantics

8.4.1 State Variables

None

8.4.2 Environment Variables

- Network connection with HTTPS support
- Access to AWS S3 endpoints and SDK

8.4.3 Assumptions

- AWS S3 credentials and configurations are valid and available.
- The network connection is stable during the upload process.

8.4.4 Access Routine Semantics

uploadData():

- transition: Sends formatted data to AWS S3 using SDK.
- output: Returns the status of the upload operation.
- exception: Throws exception for failed uploads or timeouts. specific exceptions mentioned above.

8.4.5 Local Functions

None

9 MIS of OCR Processing Module

9.1 Module

9.2 Uses

- OCR Processing Module

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
parseImage	Image from S3	Metadata formatted in JSON	AccessDeniedException, DependencyException, ImageNotFoundException

9.4 Semantics

9.4.1 State Variables

None

9.4.2 Environment Variables

- Access to AWS Lambda runtime environment

9.4.3 Assumptions

- The uploaded image is of sufficient quality for OCR processing.
- OCR tools and libraries (e.g., Tesseract) are correctly configured.

9.4.4 Access Routine Semantics

parseImage():

- transition: Analyzes the image to detect text and center points, invalid areas are skipped.

- output: Returns metadata (names and center points) extracted from the image.
- exception: Throws ProcessingException for unsupported image formats or errors during processing.

9.4.5 Local Functions

None

10 MIS of Output Storage Module

10.1 Module

10.2 Uses

- Output Storage Module
- OCR Processing Module

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
storeData	Extracted metadata, Name and center point	Success/failure response	AccessDeniedException, ValidationException, RetrievalException

10.4 Semantics

10.4.1 State Variables

None

10.4.2 Environment Variables

- Connection to dynamoDB (connection URI)

10.4.3 Assumptions

- DynamoDB schema is correctly configured.
- Data types and indexes align with the system's query requirements.

10.4.4 Access Routine Semantics

storeData():

- transition: Writes extracted data and metadata into DynamoDB.
- output: Returns the status of the storage operation.
- exception: Throws StorageException for failed write operations.

10.4.5 Local Functions

None

11 MIS of UI Parsing Module

11.1 Module

11.2 Uses

- Back-End Logic
- AWS Lambda
- Database Module (if applicable for data storage or retrieval)

11.3 Syntax

11.3.1 Exported Constants

- MAX_FILE_SIZE: Maximum allowable file size for uploads (integer, in MB)
- SUPPORTED_FORMATS: List of supported file formats (e.g., .png, .jpg, .jpeg)

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
uploadImage	imageFile (file)	uploadStatus (boolean)	FileSizeExceededError, InvalidFormatError
processImage	imageData (binary)	processedData (JSON)	ProcessingError
validateInput	imageFile (file)	isValid (boolean)	ValidationError
getFallbackData	errorContext (object)	fallbackResponse (JSON)	DataUnavailableError

11.4 Semantics

11.4.1 State Variables

- uploadedImages: Stores metadata of images uploaded by users
- processingQueue: Tracks images currently in processing

11.4.2 Environment Variables

- File System: Used for temporary storage of uploaded files
- Internet Connection: For AWS Lambda calls, if required

11.4.3 Assumptions

- Input files are uploaded through a user interface that pre-validates file formats
- AWS Lambda functions are available and operational during runtime
- The file system can handle temporary file storage without interruptions

11.4.4 Access Routine Semantics

uploadImage():

- transition: Moves the file to a secure temporary storage location.
- output: Returns true if the upload succeeds, otherwise raises an exception.
- exception: Throws FileSizeExceededError if the file size exceeds the limit. Throws InvalidFormatError if the file format is unsupported.

processImage():

- transition: Sends the image data for processing and updates the processedData variable.
- output: Returns structured JSON data containing extracted composite metadata.
- exception: Throws ProcessingError if the image cannot be processed

validateInput():

- transition: Performs checks on the input file (size, format).
- output: Returns true if the file is valid; otherwise, false.
- exception: Throws ValidationError if validation fails.

getFallbackData():

- transition: Generates or retrieves fallback data for a failed process.
- output: Returns a pre-defined or dynamically generated fallback JSON.
- exception: Throws DataUnavailableError if fallback data cannot be provided.

11.4.5 Local Functions

- logError: Logs details about errors during upload or processing for debugging and auditing purposes
- generateFallbackResponse: Creates a generic fallback response for failed processes

12 MIS of Graphical User Interface Module

12.1 Module

12.2 Uses

- React Front-End
- Input Processing Module (M12)

12.3 Syntax

12.3.1 Exported Constants

nONE

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
initializeUI	configurationSettings (object)	successMessage (string)	InvalidConfigurationError
handleUserInput	userInput (string)	processedData (object)	InputValidationError
renderUI	UIState (object)	renderStatus (boolean)	RenderFailureError
updateStyle	styleParameters (object)	updateConfirmation (string)	InvalidStyleParameterError

12.4 Semantics

12.4.1 State Variables

- currentView: String - Tracks the current page/view displayed to the user.
- userSessionData: Object - Stores temporary session data for the active user.

12.4.2 Environment Variables

- Screen: Used to render the graphical interface.
- Input Devices: Handles mouse, keyboard, and touchscreen inputs.

12.4.3 Assumptions

- React is functional and integrated with the back-end API.
- User devices support modern browsers with JavaScript enabled.

12.4.4 Access Routine Semantics

renderInterface():

- transition: Loads and displays the user interface components.
- output: None
- exception: Raises RenderingError for issues with loading components.

handleUserInput():

- transition: Processes the user event and triggers the appropriate action.
- output: Returns true if the event is processed successfully; false otherwise.
- exception: Raises InputError for invalid or unsupported inputs.

12.4.5 Local Functions

- initializeReactComponents(): Sets up and initializes React components.
- updateView(currentView): Updates the GUI view based on user interaction.

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.

13 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?

    - Everybody understood the structure of the design very well. We were all on the same page which let us complete this deliverable with confidence.

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

    - Some pain points we experienced with this deliverable were understanding how to break up and modularize our design components that we created before. We solved this by speaking with our TA and clarifying how we can divide 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?

    - The design decision to use cloud stemmed from speaking with our client and other stakeholders due to the nature of the hosting availability at McMaster. For those decisions that did not stem, they came from previous experiences in how to setup the infrastructures.

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?

    - None, if they need to we will go back to change them after getting reviewed.

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)

- Some limitations of our solution are the security and reliability. Spend more time perfecting the modularization, as well as understanding the most secure ways to setup the architecture with help from professionals. While we are secure and reliable enough now, in terms of scaling we don't know.

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)

    - Not many other design solutions were considered. The only other design solution was to directly host a database on McMaster's Servers but this came with many complications as well as setting up meetings with busy individuals who were in charge of the departments. Due to how easy setting up cloud was in comparison, we went with that route.