Module Interface Specification for Software Engineering

Team #13, ARC
Avanish Ahluwalia
Russell Davidson
Rafey Malik
Abdul Zulfiqar

January 17, 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 [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 **?**, 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 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 Sub-Realms

6.1 Module

SubRealms

6.2 Uses

Local Database Manager

6.3 Syntax

6.3.1 Exported Constants

N/A

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
getMembers	$\mathbb{N}$	Array$\langle\mathbb{N}\rangle$	SubRealmNotFound
addMember	$\mathbb{N}$, $\mathbb{N}$	-	SubRealmNotFound, UserAlreadyIn-SubRealm
removeMember	$\mathbb{N}$, $\mathbb{N}$	-	SubRealmNotFound, MemberNot-Found
createNew	Array$\langle\mathbb{N}\rangle$	$\mathbb{N}$	-
deleteExisting	$\mathbb{N}$	-	SubRealmNotFound

6.4 Semantics

6.4.1 State Variables

- $subRealms$: A mapping of $subRealmID \rightarrow subRealm$, where each $subRealm$ contains a list of $members$ (user IDs).

6.4.2 Environment Variables

- $localDB$: The local database used for storing sub-realm data temporarily.

6.4.3 Assumptions

Sub-realm IDs ($subRealmID$) are unique, and all user IDs ($userID$) are valid.

6.4.4 Access Routine Semantics

getMembers(subRealmID):

- **transition**: N/A
- **output**: Returns the list of $userIDs$ that are members of the sub-realm identified by $subRealmID$.
- **exception**:
  - $SubRealmNotFound$: If $subRealmID$ does not exist.

addMember(subRealmID, userID):

- **transition**: If $subRealmID \in subRealms$ and $userID \notin subRealms[subRealmID].members$, add $userID$ to the members of the sub-realm identified by $subRealmID$.
- **output**: N/A
- **exception**:
  - $SubRealmNotFound$: If $subRealmID$ does not exist.
  - $UserAlreadyInSubRealm$: If $userID$ is already a member of the sub-realm.

removeMember(subRealmID, userID):

- **transition**: If $subRealmID \in subRealms$ and $userID \in subRealms[subRealmID].members$, remove $userID$ from the members of the sub-realm identified by $subRealmID$.
- **output**: N/A
- **exception**:
  - $SubRealmNotFound$: If $subRealmID$ does not exist.
  - $MemberNotFound$: If $userID$ is not a member of the sub-realm.

createNew(memberList):

- **transition**: A new sub-realm is created with a unique $subRealmID$, and its $members$ are initialized to $memberList$. The new sub-realm is stored in both the local and server databases.
- **output**: Returns the unique $subRealmID$ of the newly created sub-realm.
- **exception**: None

deleteExisting(subRealmID):

- **transition**: If $subRealmID \in subRealms$, remove the sub-realm from both the local and server databases.
- **output**: N/A
- **exception**:
  - $SubRealmNotFound$: If $subRealmID$ does not exist.

6.4.5 Local Functions

- `syncWithLocalDB()`: Synchronizes sub-realm data with the local database.
- `syncWithServerDB()`: Synchronizes sub-realm data with the server database.

7 MIS of Maps

7.1 Module

Maps

7.2 Uses

Local Database Manager, Maps API (external)

7.3 Syntax

7.3.1 Exported Constants

N/A

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
getMapData	$\mathbb{N}$	MapData	LocationNotFound
addMarker	$\mathbb{N}$, Location, Details	-	-
removeMarker	$\mathbb{N}$, Location	-	MarkerNotFound
updateMarker	$\mathbb{N}$, Location, Details	-	MarkerNotFound
displayMap	$\mathbb{N}$	RenderedMap	LocationNotFound

7.4 Semantics

7.4.1 State Variables

- *markers*: A collection of markers, where each marker includes its *Location* and associated *Details*.
- *mapViews*: A mapping from $\mathbb{N}$ (view IDs) to rendered map states.

7.4.2 Environment Variables

- Access to the Google Maps API.
- Access to the local database for location details.

7.4.3 Assumptions

The Google Maps API and the local database are available and functioning properly.

7.4.4 Access Routine Semantics

getMapData(viewID):

- **transition**: N/A
- **output**: Returns $MapData$ for the $viewID$, including all markers and details for the associated location.
- **exception**: $viewID \notin mapViews$

addMarker(viewID, location, details):

- **transition**: If $viewID \in mapViews$, adds a marker to the map at *location* with the given *details*.
- **output**: N/A
- **exception**: None

removeMarker(viewID, location):

- **transition**: If $location \in markers[viewID]$, removes the marker at *location* from the map.
- **output**: N/A
- **exception**: $location \notin markers[viewID]$

updateMarker(viewID, location, details):

- **transition**: If $location \in markers[viewID]$, updates the marker at *location* with new *details*.
- **output**: N/A
- **exception**: $location \notin markers[viewID]$

displayMap(viewID):

- **transition**: Renders the map for the $viewID$, including all markers and relevant details.
- **output**: Returns $RenderedMap$, which is a visual representation of the map.
- **exception**: $viewID \notin mapViews$

7.4.5 Local Functions

- `fetchLocationDetails(Location)`: Communicates with the local database to retrieve detailed information for a given location.
- `renderMap(viewID)`: Generates a visual representation of the map for the given $viewID$ using the Google Maps API.

8 MIS of Object Interaction Module

8.1 Module

ObjectInteraction

8.2 Uses

Local Database Manager

8.3 Syntax

8.3.1 Exported Constants

N/A

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
reportObject	$\mathbb{N}$, Reason, Details	-	ObjectNotFound
reactToObject	$\mathbb{N}$, Reaction	-	ObjectNotFound
fetchReactions	$\mathbb{N}$	Array⟨Reaction⟩	ObjectNotFound
fetchReports	$\mathbb{N}$	Array⟨Report⟩	ObjectNotFound
resolveReport	$\mathbb{N}$, ResolutionDetails	-	ReportNotFound

8.4 Semantics

8.4.1 State Variables

- *objects*: A collection of AR objects, each identified by a unique $\mathbb{N}$.
- *reports*: A collection of reports associated with AR objects, including *Reason* and *Details*.
- *reactions*: A collection of user reactions, associated with specific AR objects.

8.4.2 Environment Variables

- AR objects are rendered and interactable in the environment.
- The system must have a connection to the local database for storing reports and reactions.

8.4.3 Assumptions

All AR objects are assigned unique identifiers and are interactable within the system. Users have access to a predefined set of reaction types (e.g., Like, Dislike).

8.4.4 Access Routine Semantics

reportObject(objectID, reason, details):

- **transition**: Adds a new report to *reports* for the object identified by $objectID$, with the specified *reason* and *details*.
- **output**: N/A
- **exception**: $objectID \notin objects$

reactToObject(objectID, reaction):

- **transition**: Adds a *reaction* (e.g., Like, Dislike) to *reactions* for the object identified by $objectID$.
- **output**: N/A
- **exception**: $objectID \notin objects$

fetchReactions(objectID):

- **transition**: N/A
- **output**: Returns all *reactions* associated with $objectID$.
- **exception**: $objectID \notin objects$

fetchReports(objectID):

- **transition**: N/A
- **output**: Returns all *reports* associated with $objectID$.
- **exception**: $objectID \notin objects$

resolveReport(reportID, resolutionDetails):

- **transition**: Marks the *report* identified by $reportID$ as resolved and stores the $resolutionDetails$.
- **output**: N/A
- **exception**: $reportID \notin reports$

8.4.5 Local Functions

- `validateObject(objectID)`: Ensures $objectID$ corresponds to a valid AR object in the system.
- `notifyUser(reportID)`: Sends a notification to the user who submitted the report, indicating its resolution status.

9 MIS of Local Database Manager

9.1 Module

LocalDBM

9.2 Uses

Server Database Manager

9.3 Syntax

9.3.1 Exported Constants

N/A

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
fetchData	Query	Data	DataNotFound
saveData	Key, Data	-	-
updateData	Key, Data	-	DataNotFound
deleteData	Key	-	DataNotFound
syncWithServer	-	-	ServerError
getCachedData	Key	Data	DataNotCached

9.4 Semantics

9.4.1 State Variables

- *localCache*: A local in-memory or on-disk cache, keyed by unique Key, storing frequently accessed data.
- *lastSyncTime*: A timestamp of the last successful synchronization with the server database.

9.4.2 Environment Variables

- Access to the server database for retrieving and storing persistent data.
- A local caching mechanism (e.g., in-memory cache or local storage).

9.4.3 Assumptions

The server database is available and operational for syncing, and the local caching system has sufficient storage capacity.

9.4.4 Access Routine Semantics

fetchData(query):

- **transition**: N/A
- **output**: Executes *query* on the local cache or the server database if the data is not cached, and returns the *Data*.
- **exception**: Returns $DataNotFound$ if the query does not match any records.

saveData(key, data):

- **transition**: Stores *data* in the $localCache$ with the associated *key*. Also updates the server database asynchronously.
- **output**: N/A
- **exception**: None

updateData(key, data):

- **transition**: Updates *data* in $localCache$ and synchronizes the change to the server database.
- **output**: N/A
- **exception**: $key \notin localCache$

deleteData(key):

- **transition**: Removes *data* identified by *key* from $localCache$ and the server database.
- **output**: N/A
- **exception**: $key \notin localCache$

syncWithServer():

- **transition**: Synchronizes the $localCache$ with the server database, updating any stale or missing records.
- **output**: N/A
- **exception**: $ServerError$ if the server database is unavailable or the synchronization fails.

getCachedData(key):

- **transition**: N/A
- **output**: Returns *data* from $localCache$ associated with *key*.
- **exception**: $key \notin localCache$

9.4.5 Local Functions

- `isCacheStale(key)`: Determines whether the cached data for *key* is outdated compared to the server database.
- `resolveConflict(localData, serverData)`: Resolves discrepancies between $localCache$ and server database data.

10 MIS of Server Database Manager

10.1 Module

ServerDBM

10.2 Uses

Local Database Manager

10.3 Syntax

10.3.1 Exported Constants

N/A

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
fetchData	Query	Data	DataNotFound, NetworkError
saveData	Key, Data	-	NetworkError
updateData	Key, Data	-	DataNotFound, NetworkError
deleteData	Key	-	DataNotFound, NetworkError
syncWithLocal	DataDiff	-	NetworkError

10.4 Semantics

10.4.1 State Variables

- *database*: The server database that stores all permanent data associated with the app.
- *lastSyncTime*: A timestamp indicating the last synchronization with the local database.

10.4.2 Environment Variables

- Network connectivity must be available for communication between the server database and local database manager.

10.4.3 Assumptions

The server database is accessible, operational, and synchronized with the local database manager periodically.

10.4.4 Access Routine Semantics

fetchData(query):

- **transition**: N/A
- **output**: Executes *query* on the server database and returns the corresponding *Data*.
- **exception**:
  - *DataNotFound*: If *query* does not match any records in the database.
  - *NetworkError*: If the network connection fails.

saveData(key, data):

- **transition**: Adds *data* to the server database with the associated *key*.
- **output**: N/A
- **exception**:
  - *NetworkError*: If the network connection fails.

updateData(key, data):

- **transition**: Updates *data* in the server database associated with *key*.
- **output**: N/A
- **exception**:
  - *DataNotFound*: If *key* does not exist in the database.
  - *NetworkError*: If the network connection fails.

deleteData(key):

- **transition**: Removes the record associated with *key* from the server database.
- **output**: N/A
- **exception**:
  - *DataNotFound*: If *key* does not exist in the database.
  - *NetworkError*: If the network connection fails.

syncWithLocal(dataDiff):

- **transition**: Synchronizes the *database* with changes provided in $dataDiff$ from the local database manager.

- **output**: N/A
- **exception**:
  - $NetworkError$: If the network connection fails during synchronization.

10.4.5 Local Functions

- `applyDataDiff(dataDiff)`: Applies the changes from $dataDiff$ to the server database during synchronization.
- `logSyncOperation(status)`: Logs the success or failure of the synchronization operation.

11 MIS of REST API Communication Module

11.1 Module

RESTAPICommunication

11.2 Uses

Server Database Manager, HTTP Client Library (external)

11.3 Syntax

11.3.1 Exported Constants

N/A

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
sendRequest	Endpoint, Method, Params	Response	APIError, NetworkError
parseResponse	RawResponse	ParsedResponse	ResponseParsingError
setHeaders	Headers	-	-
handleAuthentication	AuthToken	-	AuthError
checkServerStatus	-	ServerStatus	APIError, NetworkError

11.4 Semantics

11.4.1 State Variables

- *baseURL*: The base URL for the REST API server.
- *headers*: Key-value pairs for HTTP headers, including authentication tokens and content type.

11.4.2 Environment Variables

- Network connectivity for sending HTTP requests to the REST API server.

11.4.3 Assumptions

The REST API server follows standard HTTP and REST conventions, and the API endpoints are well-documented and accessible.

11.4.4 Access Routine Semantics

sendRequest(endpoint, method, params):

- **transition**: Sends an HTTP request to the API server at $baseURL + endpoint$ using the specified HTTP *method* (e.g., GET, POST, PUT, DELETE) and *params* as query parameters or request body.
- **output**: Returns the *Response* received from the API server.
- **exception**:
  - $APIError$: If the server responds with an error status code (e.g., 4xx or 5xx).
  - $NetworkError$: If the request fails due to network issues.

parseResponse(rawResponse):

- **transition**: N/A
- **output**: Converts *rawResponse* (raw HTTP response) into a structured $ParsedResponse$ (e.g., JSON or XML object).
- **exception**:
  - $ResponseParsingError$: If the *rawResponse* cannot be parsed due to invalid format.

setHeaders(headers):

- **transition**: Updates the *headers* used for subsequent HTTP requests.
- **output**: N/A
- **exception**: None

handleAuthentication(authToken):

- **transition**: Sets the authentication token in the *headers* for authorized requests.
- **output**: N/A
- **exception**:
  - $AuthError$: If the $authToken$ is invalid or rejected by the server.

checkServerStatus():

- **transition**: N/A

- **output**: Returns the *ServerStatus* indicating whether the API server is reachable and operational.
- **exception**:
  - *APIError*: If the server responds with an error status code.
  - *NetworkError*: If the request fails due to network issues.

11.4.5 Local Functions

- `buildURL(endpoint, params)`: Constructs the complete URL for the API request by appending *endpoint* to *baseURL* and encoding *params* as query parameters.
- `logRequest(requestDetails)`: Logs details of the outgoing API request for debugging purposes.
- `retryRequest(requestDetails)`: Attempts to resend a failed request based on the retry policy.

12 MIS of Object Importer Module

12.1 Module

ObjectImporter

12.2 Uses

Local Database Manager

12.3 Syntax

12.3.1 Exported Constants

N/A

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
importObject	FilePath	ObjectID	InvalidFileFormat, ImportError
validateObject	FilePath	Boolean	InvalidFileFormat
addObjectToInventory	ObjectID	-	ObjectAlreadyExists
listSupportedFormats	-	Array⟨String⟩	-

12.4 Semantics

12.4.1 State Variables

- $supportedFormats$: A list of file formats (e.g., OBJ, FBX, GLTF) that the module can process.
- $importedObjects$: A mapping of $ObjectID \rightarrow ObjectMetadata$, representing all objects imported by the user.

12.4.2 Environment Variables

- File system access for reading 3D model files.
- Network connectivity for syncing imported objects with the server database.

12.4.3 Assumptions

The file paths provided are accessible, and the objects being imported are in formats supported by the module.

12.4.4 Access Routine Semantics

importObject(filePath):

- **transition**: Reads the 3D model file from $filePath$, parses it using the 3D Model Parser Library, and generates an $ObjectID$. The object is then stored locally and synced with the server database.
- **output**: Returns the $ObjectID$ of the successfully imported object.
- **exception**:
  - $InvalidFileFormat$: If the file format is not supported.
  - $ImportError$: If the file cannot be read or parsed due to corruption or other issues.

validateObject(filePath):

- **transition**: N/A
- **output**: Returns $true$ if the file at $filePath$ is in a supported format and passes initial validation, $false$ otherwise.
- **exception**:
  - $InvalidFileFormat$: If the file format is not supported.

addObjectToInventory(objectID):

- **transition**: Adds the object identified by $objectID$ to the user's inventory and marks it as available for use within the app.
- **output**: N/A
- **exception**:
  - $ObjectAlreadyExists$: If the object is already present in the user's inventory.

listSupportedFormats():

- **transition**: N/A
- **output**: Returns the list of $supportedFormats$, indicating which file types can be imported.
- **exception**: None

12.4.5 Local Functions

- `parseFile(filePath)`: Reads and parses the 3D model file to extract metadata and geometry.
- `generateObjectID(metadata)`: Generates a unique identifier for the imported object based on its metadata.
- `syncObjectWithServer(objectID)`: Uploads the imported object's metadata to the server database.

13 MIS of Realm Interface Module

13.1 Module

RealmInterface

13.2 Uses

6, 7, 3D Renderer, AR Framework

13.3 Syntax

13.3.1 Exported Constants

N/A

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
displayObjectsInRealm	$\mathbb{N}$	-	SubRealmNotFound
navigateToMap	-	-	-
updateDisplayedObjects	$\mathbb{N}$, Array$\langle\mathbb{N}\rangle$	-	SubRealmNotFound
getActiveSubRealm	-	$\mathbb{N}$	NoActiveSubRealm
setActiveSubRealm	$\mathbb{N}$	-	SubRealmNotFound

13.4 Semantics

13.4.1 State Variables

- *activeSubRealm*: The *subRealmID* of the currently selected sub-realm.
- *displayedObjects*: A list of 3D objects being rendered in the current view.

13.4.2 Environment Variables

- 3D rendering engine for displaying AR objects.
- Maps module interface for quick navigation.

13.4.3 Assumptions

A valid *subRealmID* corresponds to an existing sub-realm, and all objects in the realm are properly loaded.

13.4.4 Access Routine Semantics

displayObjectsInRealm(subRealmID):

- **transition**: Retrieves the 3D objects associated with $subRealmID$ and displays them in the AR view.
- **output**: N/A
- **exception**:
  - $SubRealmNotFound$: If $subRealmID$ does not exist.

navigateToMap():

- **transition**: Opens the Maps interface to display the map view.
- **output**: N/A
- **exception**: None

updateDisplayedObjects(subRealmID, objectIDs):

- **transition**: Updates the list of objects displayed in the AR view for the specified $subRealmID$. Removes any previously displayed objects not in $objectIDs$.
- **output**: N/A
- **exception**:
  - $SubRealmNotFound$: If $subRealmID$ does not exist.

getActiveSubRealm():

- **transition**: N/A
- **output**: Returns the $subRealmID$ of the currently active sub-realm.
- **exception**:
  - $NoActiveSubRealm$: If no sub-realm is currently active.

setActiveSubRealm(subRealmID):

- **transition**: Sets the $subRealmID$ as the $activeSubRealm$ and updates the displayed objects accordingly.
- **output**: N/A
- **exception**:
  - $SubRealmNotFound$: If $subRealmID$ does not exist.

13.4.5 Local Functions

- `loadObjects(subRealmID)`: Loads 3D object data for the given $subRealmID$ from the local or server database.
- `renderObjects(objectList)`: Renders the provided list of 3D objects in the AR view.
- `clearDisplay()`: Removes all objects currently displayed in the AR view.

14 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?

   Our design decisions came as a natural consequence of our requirements, many of which did stem from speaking to clients. So while indirectly influenced, our design decisions were mostly made to be the easiest way to meet the requirements we set out

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)

   One of the key components of our design is the client server system for storing user data. If we had infinite resources, it would be beneficial to move more logic to the server to relieve the client from certain expensive computations. The reason our current design has them on the client side is to reduce the complexity of client server communication to simplify the implementation.

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)

In our initial solution, user content generation was meant to be internal, and more components were designed to be on the server. This design would have been ideal for users, and more maintainable in the long run, but came at an exorbiant cost for compute for object generation, and a complex design that would be infeasible to build in the alloted time. The driving force behind our changes since then were to make our app easier to implement, and less expensive to run.