4 Task 2.1

Question 4. Draw activity diagrams to capture the business process between systems and the stakeholders in Task Assignment Module.

Solution

4.1 Theoretical basis: Activity diagram

Activity diagram shows the activities in a process and the flow of control from one activity to another.

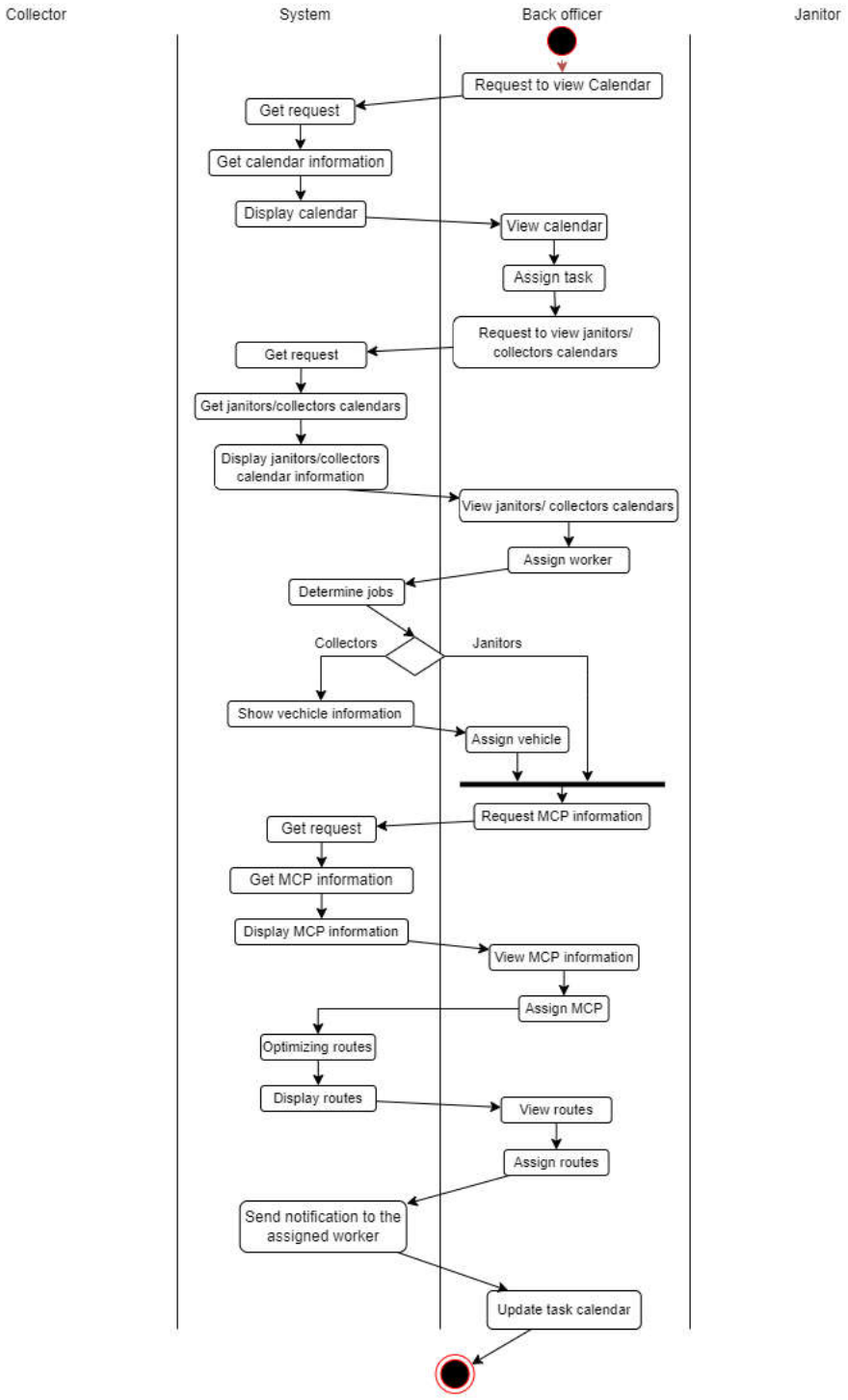
- The start of a process is indicated by a filled circle, the end by a filled circle inside another circle.
- Rectangles with round corners represent activities, that is, the specific sub-processes that must be carried out.
- Arrows represent the flow from one activity to another, and a solid bar indicates activity coordination.
- When the flow from more than one activity leads to a solid bar, then all of these activities must be complete before progress is possible.
- When the flow from a solid bar leads to a number of activities, these may be executed in parallel.
- Arrows may be annotated with guards (in square brackets) that specify when that flow is followed.

4.2 Activity diagram for Task Assignment module

For the sake of a clearer description of the Task Assignment module, our group divide the Task Assignment module into 4 smaller parts: Task Assignment, Sending Message, Viewing Task and MCP Notification.

Please follow this link for a more detailed view of these diagrams. (This link contains **4 ACTIVITY DIAGRAMS!**)

4.2.1 Activity diagram for Task Assignment


Collector
System
Back officer
Janitor
Request to view Calendar
Get request
Get calendar information
Display calendar
View calendar
Assign task
Request to view janitors/ collectors calendars
Get request
Get janitors/collectors calendars
Display janitors/collectors calendar information
View janitors/ collectors calendars
Assign worker
Determine jobs
Collectors
Janitors
Show vechicle information
Assign vehicle
Request MCP information
Get request
Get MCP information
Display MCP information
View MCP information
Assign MCP
Optimizing routes
Display routes
View routes
Assign routes
Send notification to the assigned worker
Update task calendar

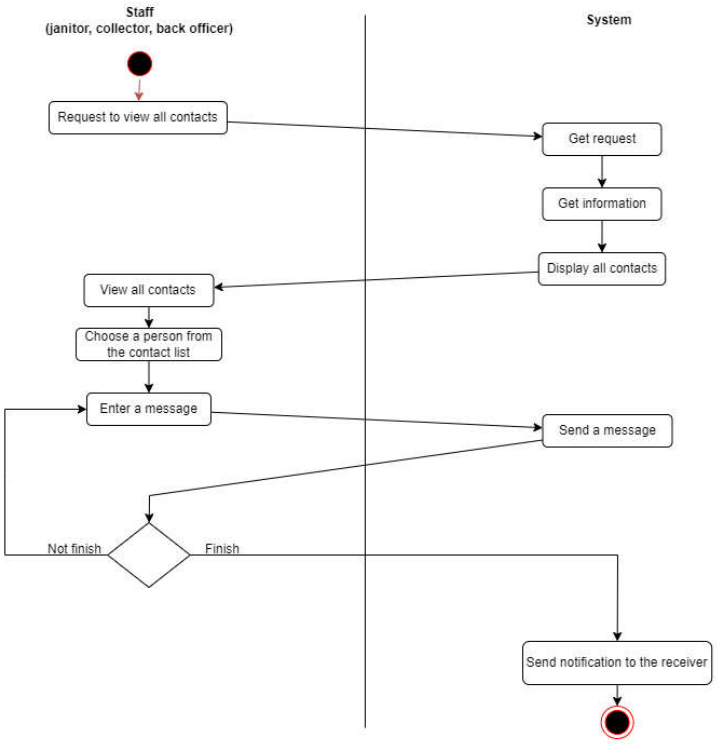
Description:

1. View calendar
    - First, the Back Officer sends a request to view the task calendar (a calendar that is noted with the upcoming tasks).
    - Second, the system receives the request from the Back Officer UI.
    - Third, the system gets the calendar information.
    - Fourth, the system displays the calendar to the Back Officer UI.
    - Fifth, hence, the Back Officer can view the task calendar.
    - Sixth, the Back Officer chooses and assigns a task.
2. View janitors and collectors working shifts
    - Seventh, the Back Officer sends a request to view the janitors and collectors working shifts in that day.
    - Eighth, the system receives the request.
    - Ninth, the system gets the needed information from the database.
    - Tenth, the system displays the information of the janitors and collectors to the Back Officer UI.
    - Eleventh, the Back Officer can view those information.
    - Twelfth, the Back Officer assigns the worker.
3. Assign tasks and vehicles
    - Thirteenth, the system determines whether the worker assigned by the Back Officer is a janitor or a collector.
    - Fourteenth, if he is a collector, then the system will show the available vehicles for the Back Officer to choose.
    - Fifteenth, the Back Officer will assign a vehicle.
4. Assign Major Collecting Points
    - Sixteenth, the Back Officer will send a request to the system to view the available MCP.
    - Seventeenth, the system receives the request.
    - Eighteenth, the system gets the MCP information from the UWC 1.0 Database.
    - Nineteenth, the system displays the available MCP to the Back Officer UI.
    - Twentieth, the Back Officer can view the MCP information.
    - Twenty-first, the Back Officer choose one MCP and assign it to the worker.

5. Assign routes

- Twenty-second, the system will use the shortest path algorithm such as Dijkstra to determine the most optional route.
- Twenty-third, the system will display the route onto the Back Officer UI.
- Twenty-fourth, the Back Officer can see the routes (can be more than one).
- Twenty-fifth, the Back Officer will choose one of the most optimal routes to assign to the janitors and collectors.
- Twenty-sixth, the system will display the assigned task to the assigned worker with corresponding information: vehicle, MCP, route.
- Twenty-seventh, the Back Officer may update the task calendar (cross out the latest assign task).
- Twenty-eighth, the activity diagram reaches the end symbol.

4.2.2 Activity diagram for sending message

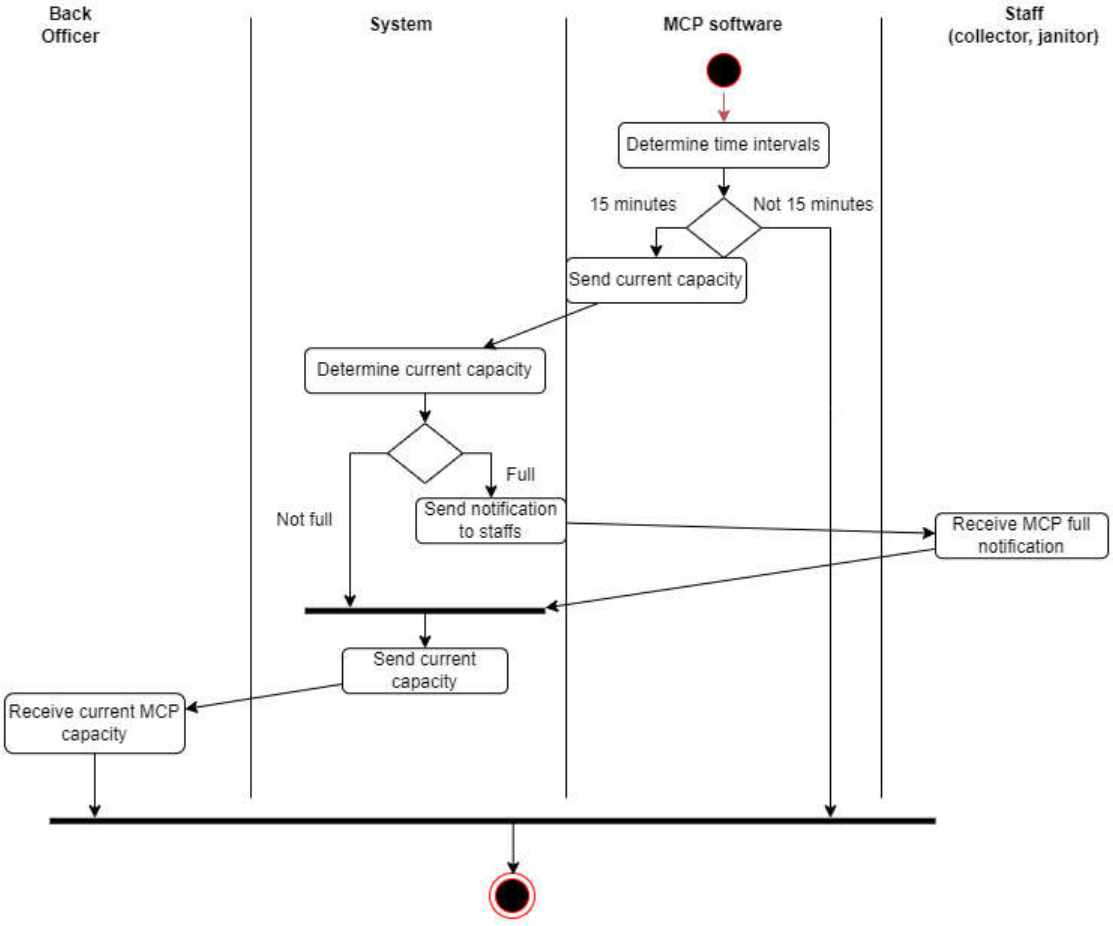


Description:

- First, the staff (either the collector, the janitor or the back officer) request to view all his available contacts.
- Second, the system gets the request.

- Third, the system gets the information from the Database provided.
- Fourth, the system displays all his available contacts onto the user UI.
- Fifth, the staff can now view all his contacts.
- Sixth, the staff then chooses another staff that he wants to communicate.
- Seventh, the staff continuously enters messages until he finishes doing so. The system sends all messages written to the receiver.
- Eighth, the system will also notice the receiver that he has coming messages from another person.

4.2.3 Activity diagram for MCP management

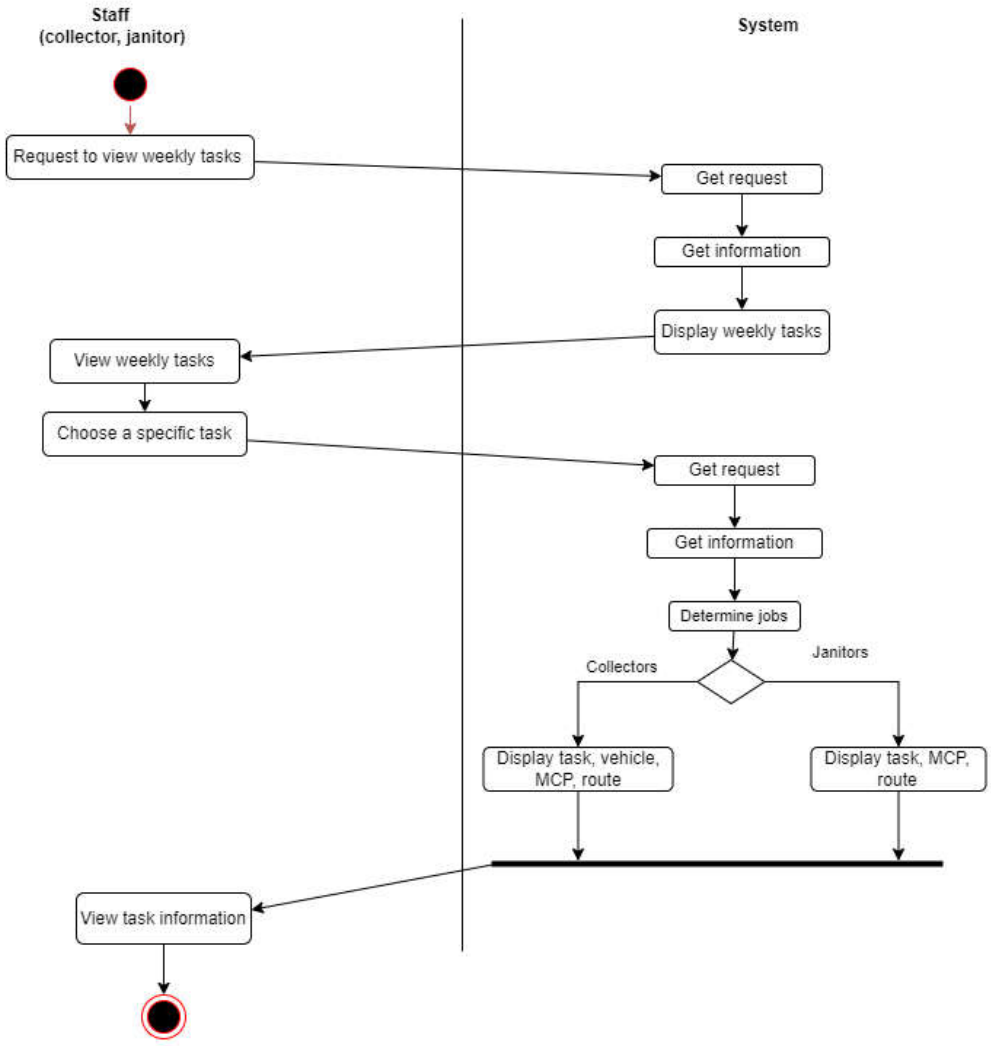


Description:

- First, the embedded software in each MCP will determine the time intervals whether it has been 15 minutes since the last update or not.

- Second, if it is, then the MCP software will send the current capacity to our system. If it is not 15 minutes, stops.
- Third, when our system receive the current capacity, it will determine whether the current capacity is full or not.
- Fourth, if it is not, then skip this step. Otherwise, send the notification to the staff about with the information that the MCP is full.
- Fifth, our system is also responsible for sending the current capacity (whether full or not) to the back officer.
- Sixth, the back officer will receive the current MCP capacity.

4.2.4 Activity diagram for viewing task



Description:

- First, the staff (either the collector or the janitor) will request to view the weekly tasks.
- Second, the system gets the request.
- Third, the system gets the information from the Database.
- Fourth, the system displays the weekly tasks to the user.
- Fifth, the user can view his weekly tasks.
- Sixth, the user may choose one particular task to see detailed information.
- Seventh, the system gets the request.
- Eighth, the system gets the information from the Database.
- Ninth, the system then determines whether the person who sends the request is a collector or a janitor.
- Tenth, if he is a collector, let him view the task with detailed information of vehicle, MCP and route. Otherwise if he is a janitor, let him view the task, MCP and route.

Note: Each MCP has an embedded system to determine the current capacity of the MCP. And we will let this software handles the jobs of delivering the current capacity back to our main system too. Otherwise, letting the Back Officer to manually walk to the MCP and determine the capacity of each MCP is totally impractical: the time he travels between MCP can be more than 15 minutes and he have to work non-stop, cannot even have lunch!

5 Task 2.2

Question 5. Propose a conceptual solution for the route planning task and draw a sequence diagram to illustrate it.

Solution

5.1 Theoretical basis

5.1.1 Conceptual design

Conceptual design is an initial/starting phase in the process of planning, during which the broad outlines of function and sort of something are coupled. It tells the customers that what the system actually will do and shows the conceptual model of the system, in other words, what the system should look like.

5.1.2 Sequence diagram

UML Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. Sequence Diagrams are time focus and they show the order of the interaction visually by using the vertical axis of the diagram to represent time what messages are sent and when. Sequence Diagrams captures:

- the interaction that takes place in a collaboration that either realizes a use case or an operation (instance diagrams or generic diagrams)
- high-level interactions between user of the system and the system, between the system and other systems, or between subsystems (sometimes known as system sequence diagrams)

A sequence diagram contains the following components:

- **Actor:** a type of role played by an entity that interacts with the subject, external to the subject, representing roles played by human users, external hardware, or other subjects.
- **Lifeline:** represents an individual participant in the Interaction.
- **Activation:** a thin rectangle on a lifeline, which represents the period during which an element is performing an operation.
- **Message:** defines a particular communication between Lifelines of an Interaction
  - **Call message:** represents an invocation of operation of target lifeline.
  - **Return message:** represents the pass of information back to the caller of a corresponded former message.
  - **Self message:** represents the invocation of message of the same lifeline.

  - **Recursive message:** represents the invocation of message of the same lifeline. It's target points to an activation on top of the activation where the message was invoked from.
  - **Create message:** represents the instantiation of (target) lifeline.
  - **Destroy message:** represents the request of destroying the lifecycle of target lifeline.
  - **Duration message:** shows the distance between two time instants for a message invocation.

- **Sequence fragments:** represented as a box, called a combined fragment, which encloses a portion of the interactions within a sequence diagram. Seven types: ref, assert, loop, break, alt, opt, neg.

5.2 Our solution for the question

5.2.1 Conceptual solution for the route planning task

Our proposed conceptual solution for the route planning task of UWC 2.0 system can be described as follow:

1. The back officer initiates the route assignment module.
2. The route assignment module calls the database to get the list of available collectors. The list of collectors is returned and displayed to the back officer's UI. If there are no collectors available, the UI will display the message "No collectors available".
3. If there are available collectors, the back officer chooses a collector to assign MCPs and create route. The module calls the database to get the list of available MCPs. The list of available MCPs is returned and displayed to the back officer's UI.
4. The back officer selects MCPs to assign to the chosen collector.
5. The system creates the most optimal route from the list of designated MCPs from the previous step.
6. The system asks for confirmation from the back officer. After receiving the confirmation from the back officer, the system will update assigned MCPs and route of the chosen collector in the database.

5.2.2 Sequence diagram of route planning task

Note: [MCP_ID] refers to an array of IDs of MCPs. It means that the back officer has chosen multiple MCPs.

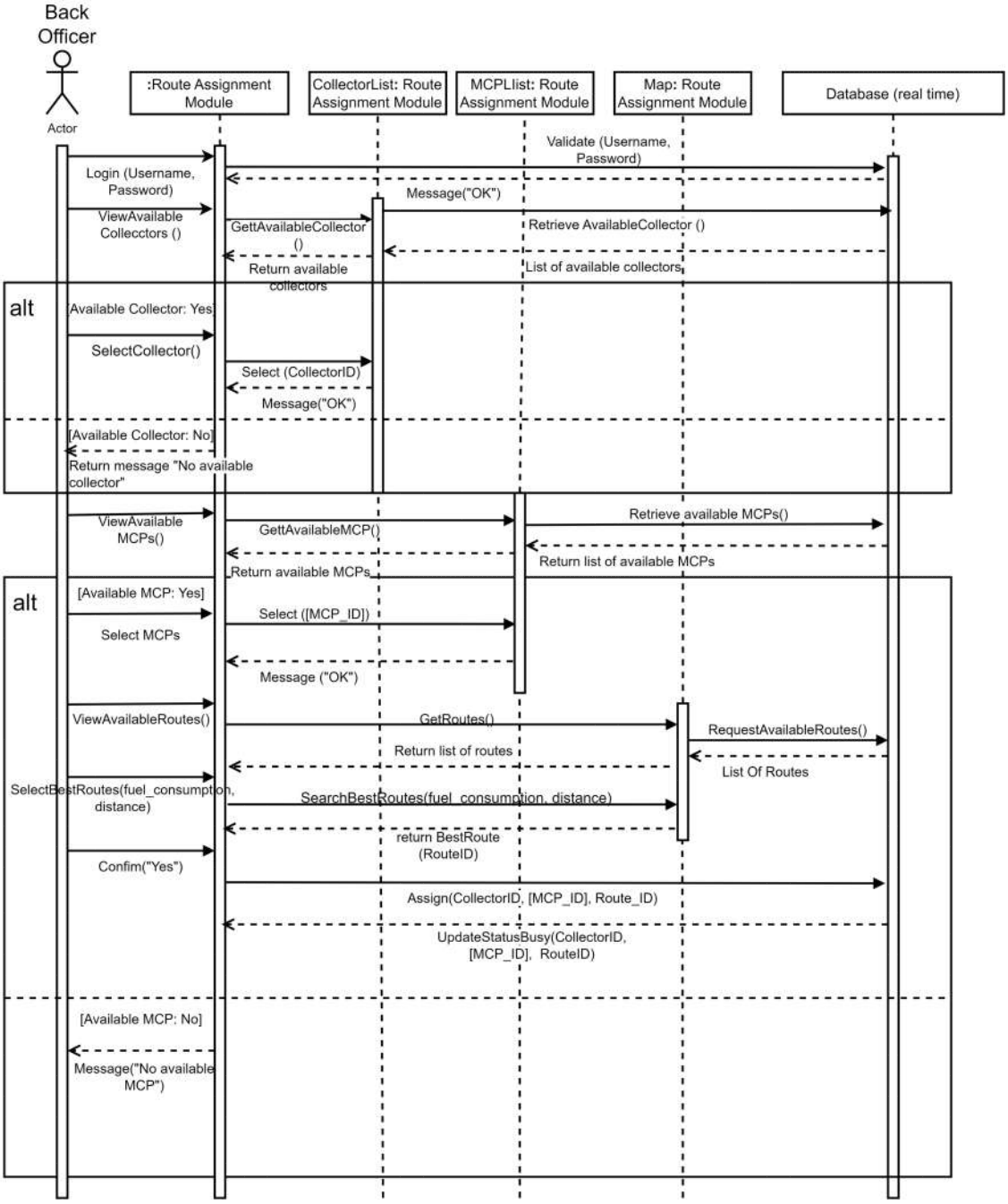


Figure 2: Sequence Diagram for the route planning process

The figure can be digitally accessed using this link

6 Task 2.3

Question 6. Draw a class diagram of Task Assignment module as comprehensive as possible

Solution

6.1 Theoretical basis: Class diagram

In UML, class diagrams are one of six types of structural diagram. Class diagrams are fundamental to the object modeling process and model the static structure of a system. Depending on the complexity of a system, we can use a single class diagram to model an entire system, or we can use several class diagrams to model the components of a system.

Class diagrams are the blueprints of the system or subsystem. We can use class diagrams to model the objects that make up the system, to display the relationships between the objects, and to describe what those objects do and the services that they provide.

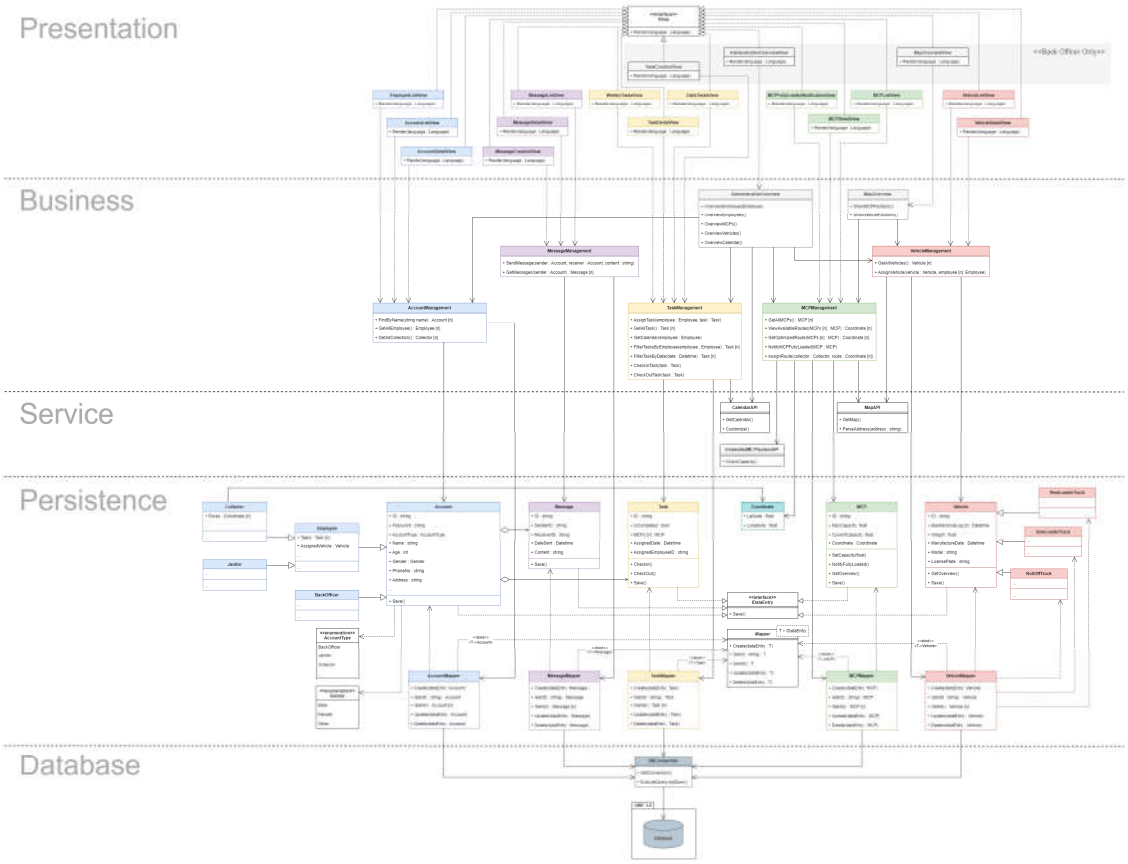
Popular model elements of a class diagram:

- **Classes:** In UML, a class represents an object or a set of objects that share a common structure and behavior. Classes, or instances of classes, are common model elements in UML diagrams. The structure of a class in class diagram consists of: class's name, attributes (optional) and operations (optional).
- **Objects:** In UML models, objects are model elements that represent instances of a class or of classes. We can add objects to our model to represent concrete and prototypical instances. A concrete instance represents an actual person or thing in the real world.
- **Packages:** Packages group related model elements of all types, including other packages.
- **Enumerations:** In UML models, enumerations are model elements in class diagrams that represent user-defined data types. Enumerations contain sets of named identifiers that represent the values of the enumeration. These values are called enumeration literals.
- **Data types:** In UML diagrams, data types are model elements that define data values. We typically use data types to represent primitive types, such as integer or string types, and enumerations, such as user-defined data types.
- **Subsystems:** A subsystem groups elements that together provide services such that other elements may access only those services and none of the elements themselves. And while packages allow us to partition our system into logical groups and relate these logical groups, subsystems allow us to consider what services these logical groups provide to one another.

Some concepts that are widely used in class diagram:

- **Dependencies:** A dependency from a source element (called the client) to a target element (called the supplier) indicates that the source element uses or depends on the target element; if the target element changes, the source element may require a change.
- **Realizations:** A realization from a source element (called the realization element) to a target element (called the specification element) indicates that the source element supports at least all the operations of the target element without necessarily having to support any attributes or associations of the target element.
- **Generalizations:** A generalization between a more general element and a more specific element of the same kind indicates that the more specific element receives the attributes, associations and other relationships, operations, and methods from the more general element.
- **Aggregation:** Aggregation is whole-part relationship between an aggregate, the whole, and its parts. This relationship is often known as a *has-a* relationship.
- **Composition:** Composition, also known as composite aggregation, is a whole-part relationship between a composite (the whole) and its parts, in which the parts must belong only to one whole and the whole is responsible for creating and destroying its parts when it is created or destroyed. This relationship is often known as a *contains-a* relationship.
- **Association ends:** An association end is an endpoint of the line drawn for an association, and it connects the association to a class. An association end may include any of the following items to express more detail about how the class relates to the other class or classes in the association:
  - Role name.
  - Navigation arrow.
  - Multiplicity specification.
  - Aggregation or composition symbol.
  - Qualifier.

6.2 Class diagram of Task Assignment module



Follow this link for a PNG with better resolution, or this link for the diagram file.

Explanation: This class diagram uses the Layered Architecture approach with 5 different layers. Every layer is made up from various components, each handles or assists a specific task of task assignment module. To fully describe our class diagram, we will use 2 different approaches for explanation:

- **Horizontal approach:** In this approach, we will provide a brief description of each layer.

    1. **Database layer:** This layer is used to store the data for the system. The database will store the data and the DBConnection class will connect this layer to persistence layer for retrieving and updating data.
    2. **Persistence layer:** Business layer and database layer are not allowed to communicate directly due to the principle of layered architecture. This layer facilitates the interaction between business layer and the database by retrieving data from database and passing them to business layer as well as sending requests from business layer to database layer.

3. **Service layer:** This layer will be the place to store external APIs like map API, which provides an interface for working with map, and use them to establish a connection between components of the business layer and outside resources such as map, MCP system, calendar, etc. This connection allows components in business layer get use of such resources, thus greatly reduce their workload and difficulty in implementation.
4. **Business layer:** This layer is the core of the whole system. It contains all the logic behind the operation of task assignment module. Business layer plays a role as a "function", it receives requests from users, processes them based on the logic and produces the desirable outputs.
5. **Presentation layer:** This layer's responsibility is to display information on the users' screen based on the types of users and types of contents that the users wish to observe. Furthermore, it also has the ability to change displayed language based on users' preference.

- **Vertical approach:** As we can see in the diagram, components relating to each other can be grouped, with each group has its own functionalities. We color-coded the groups for better visualization. In this approach, we will explain the role of each group of components:

  1. **Account system:** This system deals with managing all accounts in the UWC 2.0 software. The managing process can be described as follow: First of all, business layer receives requests related to users' account. It then passes the requests to appropriate components. In this case, AccountManagement will handle the requests based on its logic. When invoked, AccountManagement will send a request to AccountMapper to retrieve or update data in database. If retrieving data is the case, the data are then put in the Account class to be encapsulated in an Account object before sent back to AccountManagement. The data are then processed and displayed on the screen using relevant classes in presentation layer if necessary.
  2. **Message system:** This system is responsible for processing messages (creating, sending, receiving, etc). Its managing mechanism is almost the same as that of account system except for one thing: Message class gets data from both MessageMapper and Account class (to indicate sender and receiver).
  3. **Task system:** This system is specified for tasks handling and is similar in management process to message system. It also take advantage of calendar API to work with calendar.
  4. **Coordinate:** This class is used to stored the coordinate of MCPs retrieved from the database and is used in MCP managing process.
  5. **MCP system:** This system is responsible for working with MCPs' data. It managing mechanism is almost the same as that of account system except for one thing: MCPManagement class gets data from both MCPMapper and Coordinate class (to determine MCPs' location). It also uses APIs from service layer to interact with map and sensors in each MCP.

6. **Vehicle system:** This system focuses on working with vehicles' data and has analogous mechanism to account system. It also uses map API to work with the map.
7. **Overview system:** Once it receives requests, it invokes corresponding management systems and APIs to get the overview data.
8. **APIs:** This group includes MapAPI, CalendarAPI and EmbededMCPSystemAPI. These API provide the interface to work with external resources like map, calendar or sensors in MCPs.
9. **DBConnection:** This class plays a role as the middleman between persistence and database layer. It receives requests from mapper classes and provide appropriate services to work with the database.