Assignment 6

PREPARE AND IMPLEMENT SEQUENCE MODEL

Aim: Identify at least 5 major scenarios (sequence flow) for your system. Draw Sequence Diagram for every scenario by using advanced notations using UML2.0. Implement these scenarios by taking reference of design model implementation using suitable object-oriented language.

Problem Statement:

- Prepare Sequence Model
- Identify at least 5 major scenarios (sequence flow) for DEFINITION your system. Draw Sequence Diagram for every scenario by using advanced notations using UML2.0
- Implement these scenarios by taking reference of design model implementation using a suitable object-oriented language

Objective:

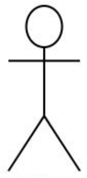
- To study and use communication.
- Draw sequence diagram
- To implement sequence diagram

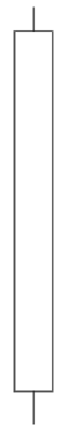
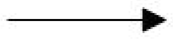
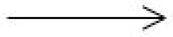
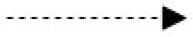
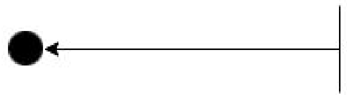
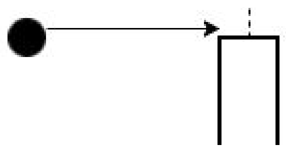
1. Relevant Theory:

1.1 Sequence Diagram

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

1. 2 Sequence Diagram Notations

Message Type	Description	Symbol
Actors	An actor in a UML diagram represents a type of role where it interacts with the system and its objects. It is important to note here that an actor is always outside the scope of the system we aim to model using the UML diagram.	

Lifelines	A lifeline is a named element which depicts an individual participant in a sequence diagram. So basically each instance in a sequence diagram is represented by a lifeline. Lifeline elements are located at the top in a sequence diagram. The standard in UML for naming a lifeline follows the following format – Instance Name : Class Name	
Synchronous messages	Synchronous messages wait for a reply before the interaction can move forward. The sender waits until the receiver has completed the processing of the message. The caller continues only when it knows that the receiver has processed the previous message i.e. it receives a reply message.	
Asynchronous Messages	An asynchronous message does not wait for a reply from the receiver. The interaction moves forward irrespective of the receiver processing the previous message or not. We use a lined arrow head to represent an asynchronous message.	
Reply Message	Reply messages are used to show the message being sent from the receiver to the sender. We represent a return/reply message using an open arrowhead with a dotted line. The interaction moves forward only when a reply message is sent by the receiver.	
Lost Message	A Lost message is used to represent a scenario where the recipient is not known to the system. It is represented using an arrow directed towards an end point from a lifeline. For example: Consider a scenario where a warning is generated.	
Found Message	A Found message is used to represent a scenario where an unknown source sends the message. It is represented using an arrow directed towards a lifeline from an endpoint. For example: Consider the scenario of a hardware failure.	

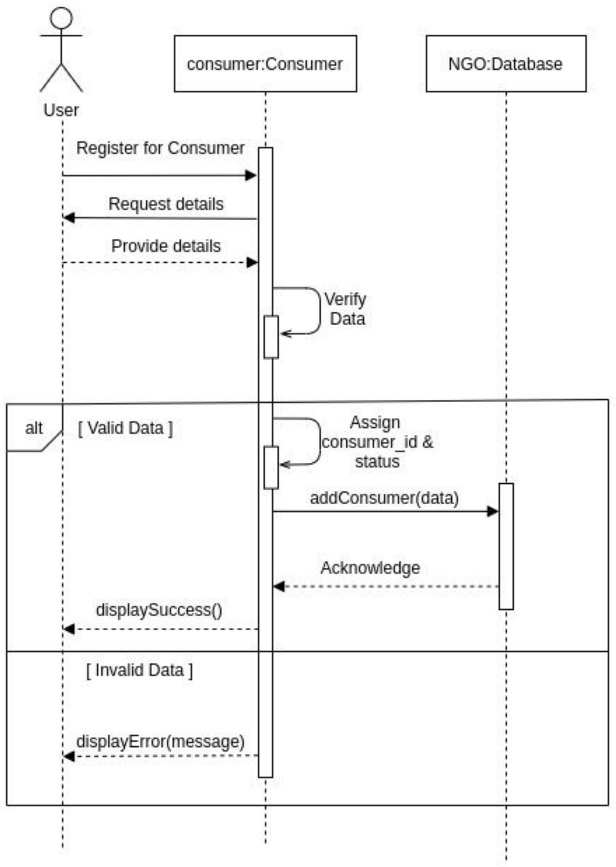


Fig 1.1 Consumer Registration

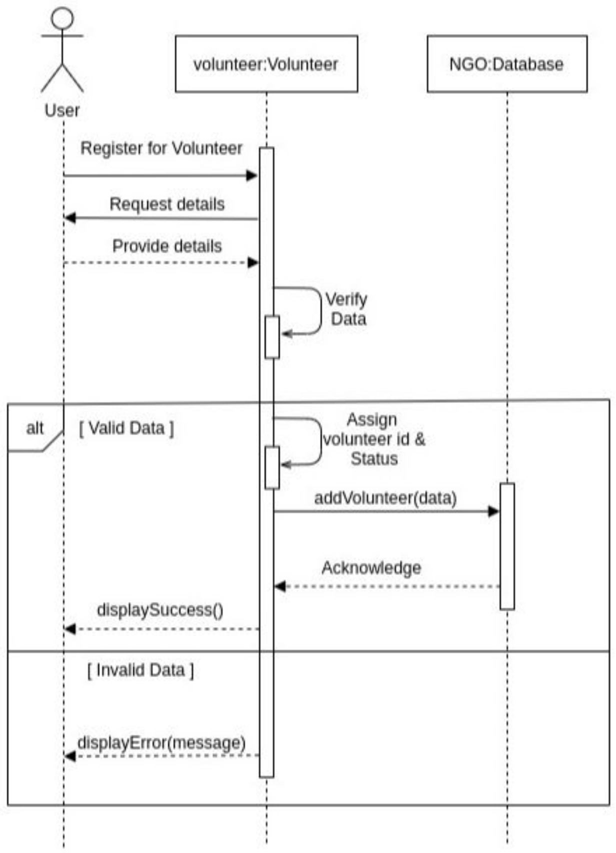


Fig 1.2 Volunteer Registration

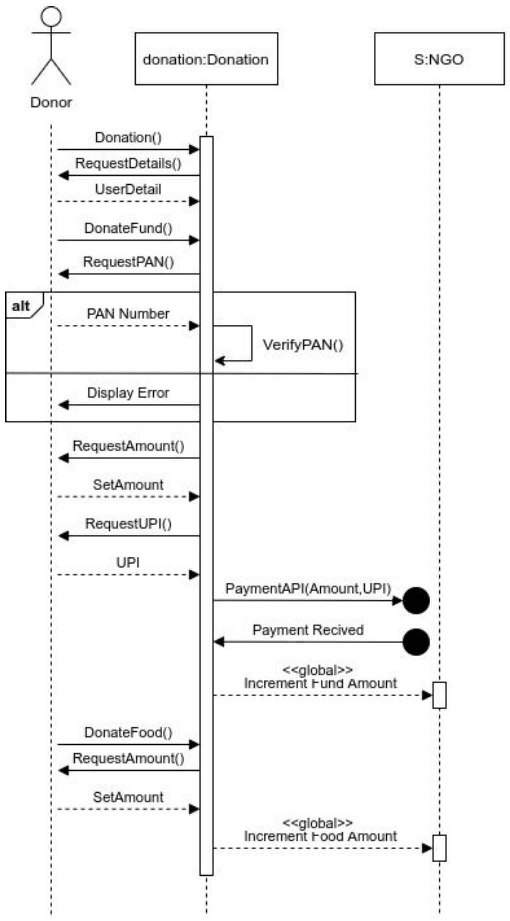


Fig 1.3 Donation

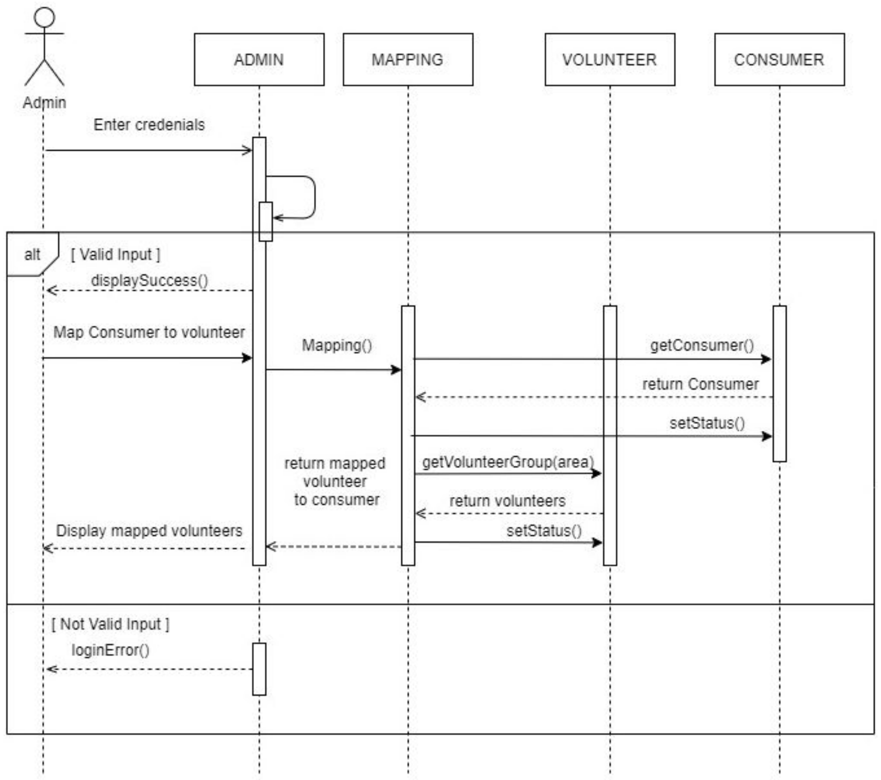


Fig 1.4 Mapping

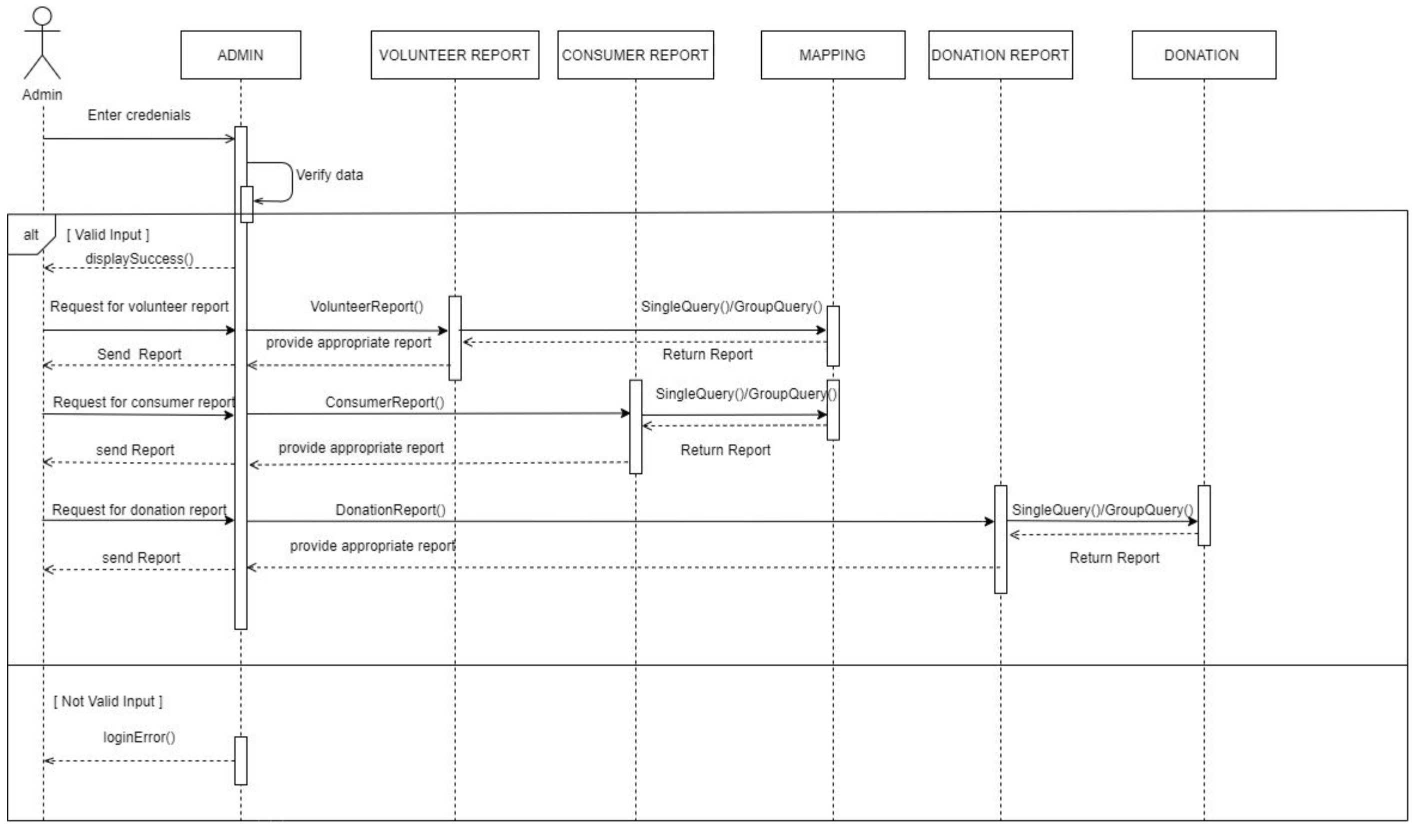


Fig 1.5 Report

2. Implementation

A. Volunteer and Consumer Registration:

Consumer Registration

i. As shown in Fig 1.1, Users who want to register as Consumer will navigate to registration for consumer. Consumer object will request the details of the user and it also handles the verification of inputted data.

ii. If the data will get validated according to policies, Consumer object will assign a consumer_id which will be used to uniquely identify each Consumer in NGO. The status shows whether Consumer is available or not at present time.

iii. After assigning consumer_id and status to Consumer, the data will get added to the database through addConsumer() method. When data of the user will get added successfully, an acknowledgement is sent to the Consumer object as well as to the user through displaySuccess() method.

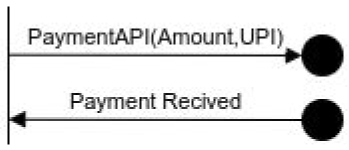
iv. If the inputted data is invalid or in missing form, then User will get an error with a message showing the details of the error through displayError() method.

Volunteer Registration

i. As shown in Fig 1.2, Users who want to register as Volunteer will navigate to registration for volunteer. Volunteer object will request the details of the user and it also handles the verification of inputted data.
ii. If the data will get validated according to policy of volunteer, Volunteer object assign a unique volunteer_id which will be used to identify each Volunteer in NGO. The status shows whether Volunteer is available or not at present time.
iii. After assigning volunteer_id and status to Volunteer, the data will get added to the database through addVolunteer() method. When data of the user will get added successfully, an acknowledgement is sent to the Volunteer object as well as to the user through displaySuccess() method.
iv. If the inputted data is invalid or in missing form then, User will get an error with a message showing the details of the error through displayError() method.

B. Donation :

i. As shown in Fig 1.3, User who want to donate fund or food to the NGO, will be able to do so by interacting with donation
ii. Donation object contain transaction between user and database, Its primary job is to accept fund and food
iii. During Donation of fund, Following action are performed
    - PAN number verification
    - The amount will be requested through payment gateway, which is third party API, thus lost and found is used



    - Once the payment is successful. The sum will be added to the database NGO fund. And the donation will be logged.

iv. During Donation of food, The sum will be added to the database NGO food

C. Mapping :

i. As Shown in Fig 1.4, Admin who wants to map volunteers to consumers, will be able to do so by interacting with the mapping class.

ii. Admin will first request mapping to admin class then admin class will start the mapping process. It will call a mapping function from the mapping class.

iii. To perform mapping, it requires a consumer which you have to select consumer and volunteers that are to be mapped. It gets the consumer from consumer class through getConsumer() method and updates it’s status to mapped.

iv. After consumer, it fetches volunteers from volunteer class through getVolunteerGroup() method according to volunteer policies and updates their status to mapped.

v. Once mapping is done, Mapping class returns mapped volunteers to admin class. Admin class then displays mapped volunteers and a particular consumer

D. Report:

i. As Shown in Fig 1.5, Admin who want to see the reports, will be able to see by interacting with the different report classes.

ii. Admin will first request to the Admin class then Admin class will interact with respective report class depending on the type of request. Such as if admin has requested for a volunteer's report admin class will interact with the volunteer_report class and if admin requests for consumer report then admin class will interact with consumer_report class and same goes for the donation report.

iii. Then these volunteer_report and consumer_report classes will interact with the mapping class for getting individual or group reports as per admin’s request.

iv. Then mapping class sends the report to the previous classes and then these classes carry the report to the admin.

Conclusion

Thus in this Assignment we have successfully Identified and implemented sequence diagrams for 5 major scenarios of our system.