IN9130 - Unassailable IT-systems

Obligatory Exercise I

An Ngoc Lam

September 19, 2018

Question I

Figure 1 illustrates the class diagram of the Product Monitoring usecase specified in the requirement.

- According to the *law of proximity*, the **Temperature**, **Humidity** and **Vibration** sensors which belong to the same **Sensor Unit** are all grouped to **Sensor** type. Similarly, **Client Factory** and **Customer Factory** are all **Factory**. Also, **Mobile Gateway** and **Static Gateway** are all **Gateway**.

- **Chemical** could be produced by many **Client Factory** and purchased by several **Customer Factory**.

- A **Container** could store various type of **Chemical** and contain only one **Sensor Unit**. It could be either stored in one **Short-term Storage** or being transported by one **Truck**.

- A **Shipment** may contain many **Containers**, involve more than one **Truck** and be delivered to one **Customer Factory**.

- There is no *construct deficit* since the class diagram in figure 1 already covers all the important objects mentioned in the requirement as well as the relationship between them. The attributes (which are mentioned in the requirement) are also included in corresponding classes.

- There is no *construct redundancy* since each actor or object specified in the requirement is represented by only one class in the diagram. Also, similar concepts are grouped together with generalization class. Similarly, *construct overload* is also avoided since there is no class represent multiple concepts.

- There is no *construct excess* since all the classes in the diagram only represent actors or objects which are explicitly mentioned in the requirement.

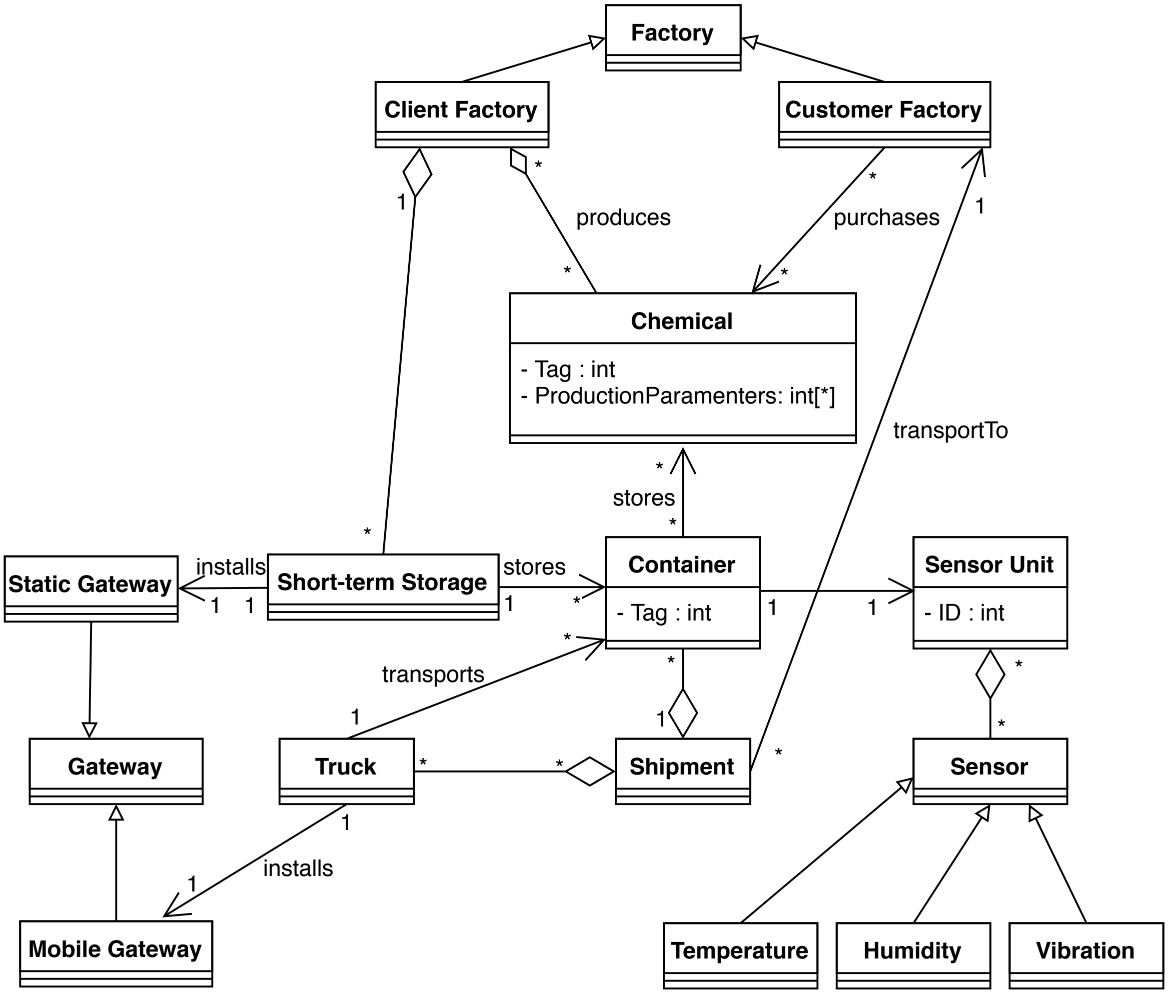
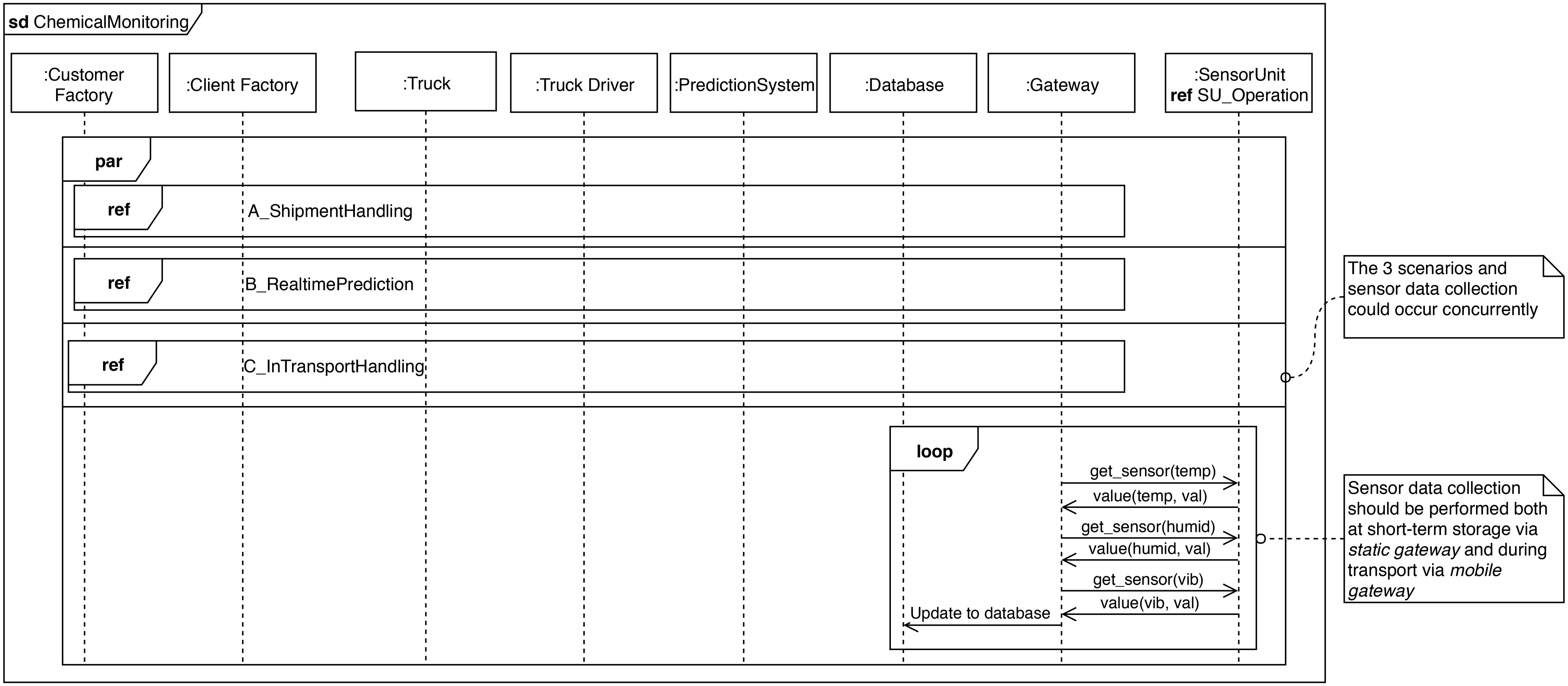


Figure 1: Class diagram of Product Monitoring

Question II

Figure 2 represents the overview of the three scenarios A, B and C. Each of these scenarios is illustrated in more details in figure 3, 4 and 5 respectively.



The 3 scenarios and sensor data collection could occur concurrently

Sensor data collection should be performed both at short-term storage via *static gateway* and during transport via *mobile gateway*

Figure 2: The overview sequence diagram

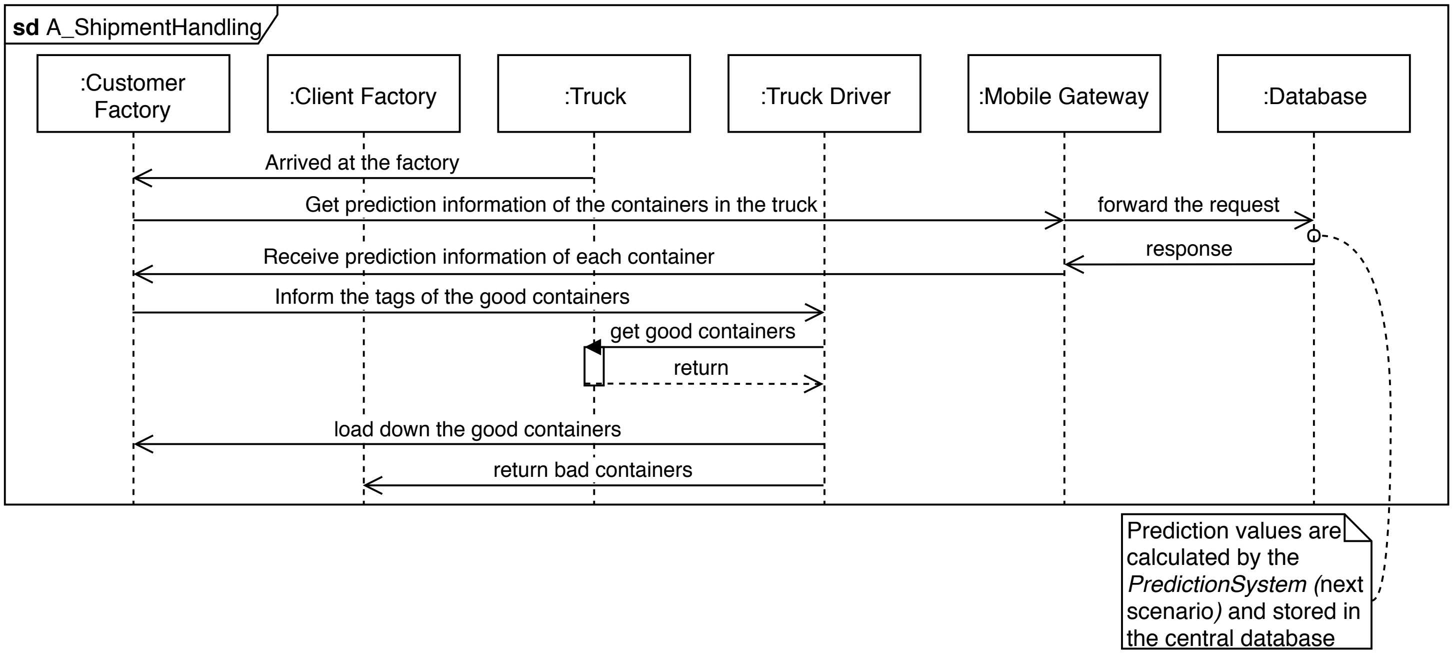

sd A_ShipmentHandling
:Customer Factory
:Client Factory
:Truck
:Truck Driver
:Mobile Gateway
:Database
Arrived at the factory
Get prediction information of the containers in the truck
forward the request
response
Receive prediction information of each container
Inform the tags of the good containers
get good containers
return
load down the good containers
return bad containers
Prediction values are calculated by the *PredictionSystem (*next scenario*)* and stored in the central database

Figure 3: Diagram for scenario A

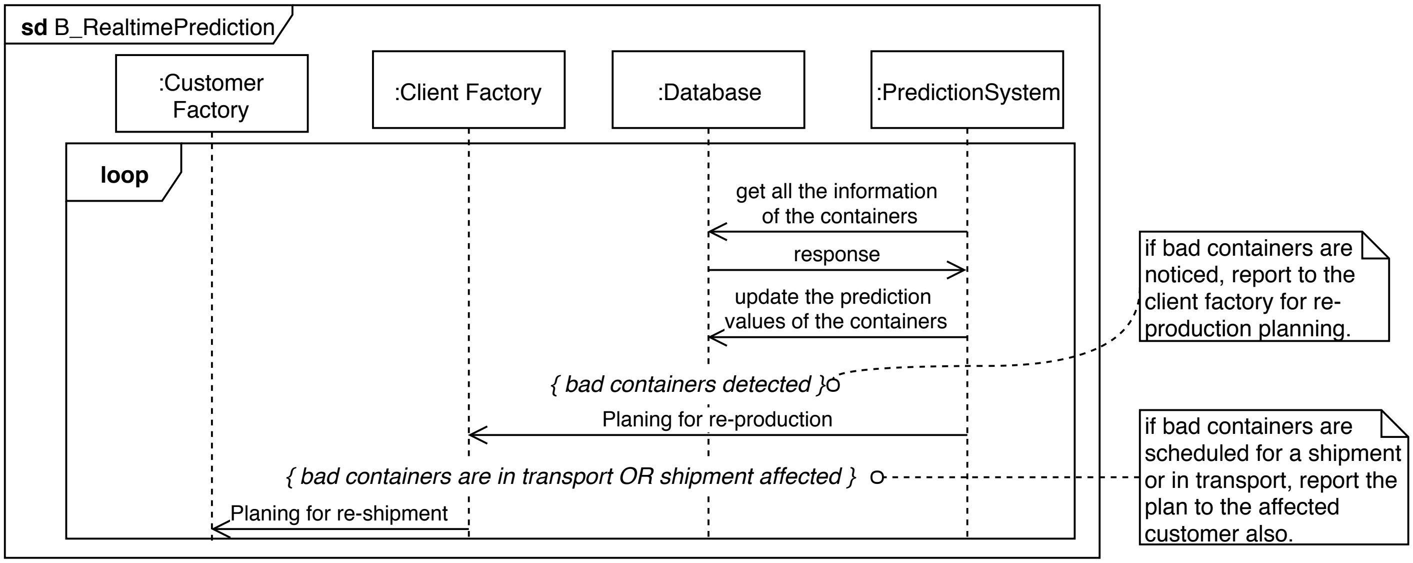

sd B_RealtimePrediction
:Customer Factory
:Client Factory
:Database
:PredictionSystem
loop
get all the information of the containers
response
update the prediction values of the containers
{ bad containers detected }
Planing for re-production
{ bad containers are in transport OR shipment affected }
Planing for re-shipment
if bad containers are noticed, report to the client factory for re-production planning.
if bad containers are scheduled for a shipment or in transport, report the plan to the affected customer also.

Figure 4: Diagram for scenario B

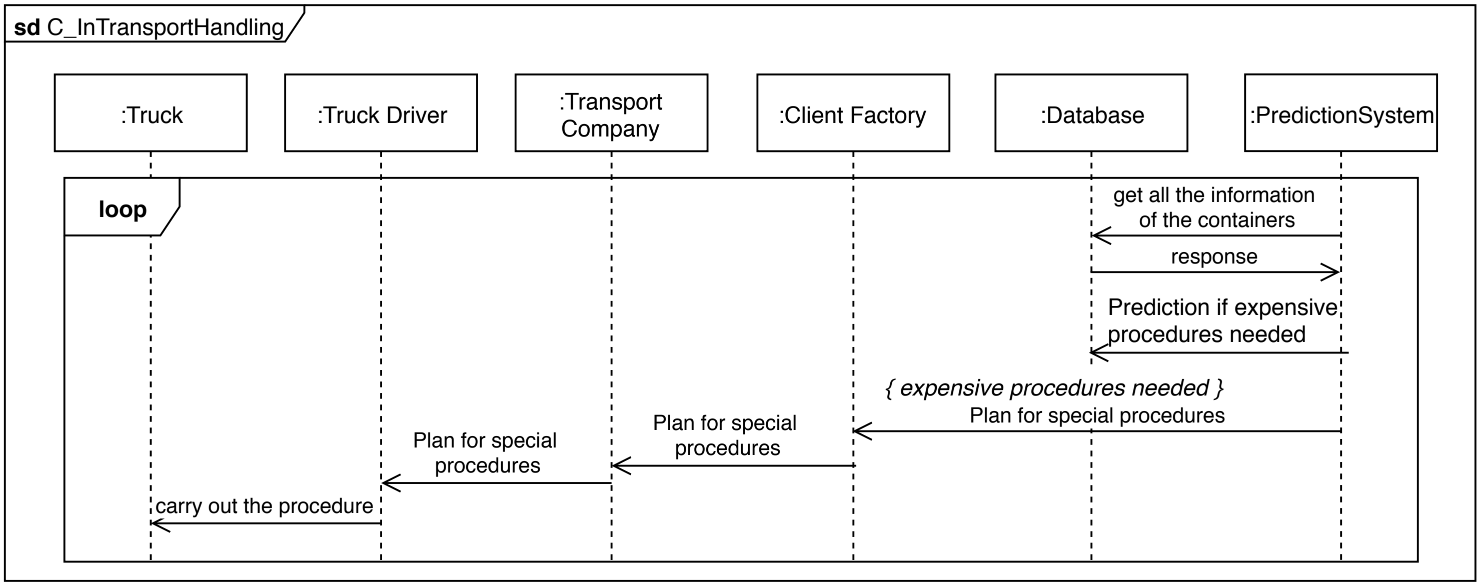

sd C_InTransportHandling
:Truck
:Truck Driver
:Transport Company
:Client Factory
:Database
:PredictionSystem
loop
get all the information of the containers
response
Prediction if expensive procedures needed
{ expensive procedures needed }
Plan for special procedures
Plan for special procedures
Plan for special procedures
carry out the procedure

Figure 5: Diagram for scenario C

Question III

Figure 6 illustrates the composite structure of the **Sensor Unit**. Figure 7 presents the internal communication of the components inside the Sensor Unit which is also referenced by the sequence diagram in figure 2.

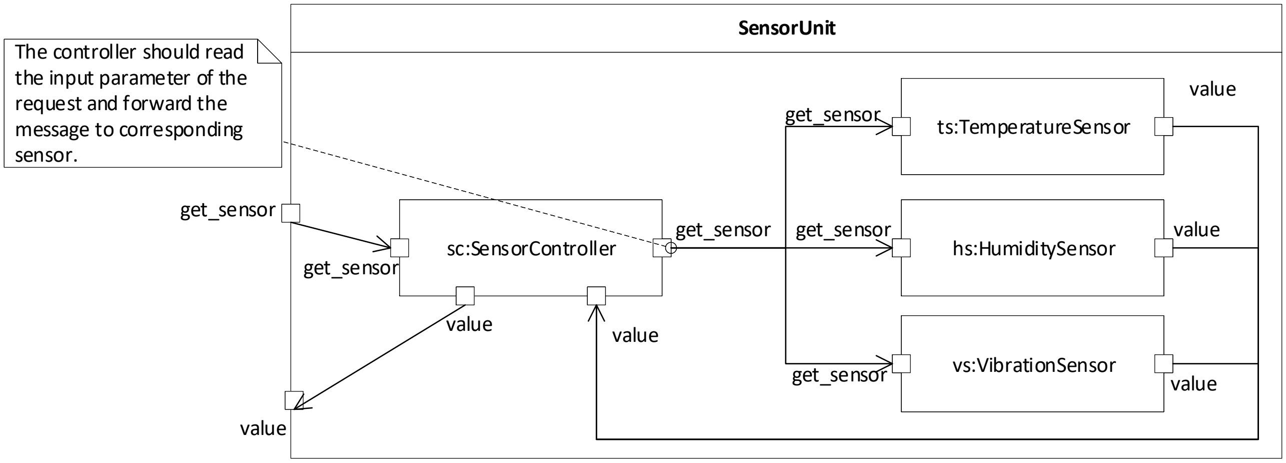


Figure 6: Composite structure of the Sensor Unit

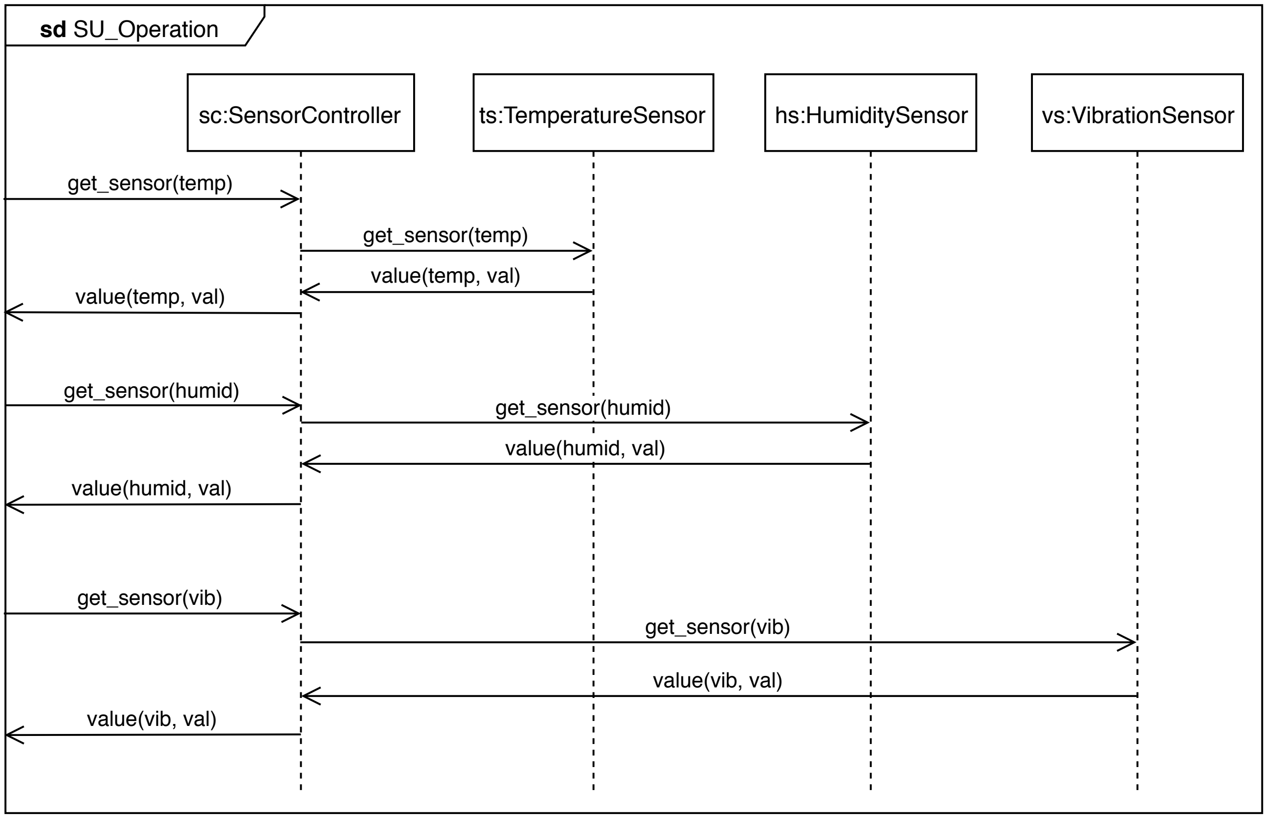


Figure 7: Sequence diagram of the Sensor Unit

Question IV

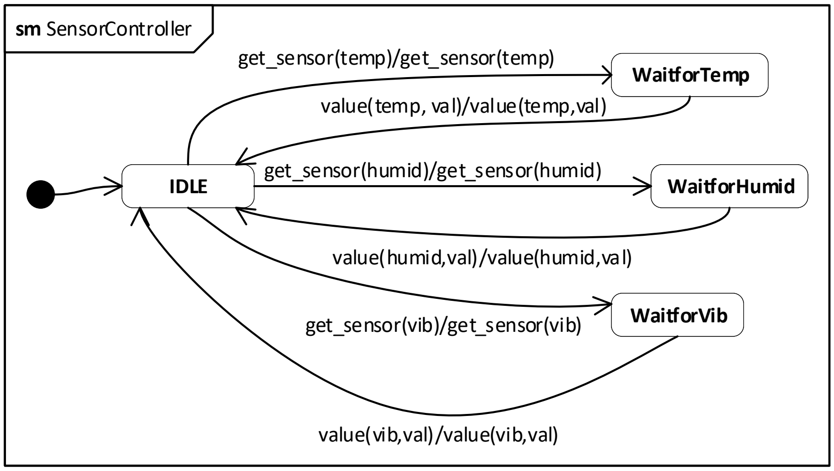

sm SensorController
get_sensor(temp)/get_sensor(temp)
WaitforTemp
value(temp, val)/value(temp,val)
IDLE
get_sensor(humid)/get_sensor(humid)
WaitforHumid
value(humid,val)/value(humid,val)
WaitforVib
get_sensor(vib)/get_sensor(vib)
value(vib,val)/value(vib,val)

Figure 8: State machine of the Sensor Controller

Figure 8 represents the behavior state machine of the **Sensor Controller** corresponding to the sequence diagram in figure 7. As can be seen from these figures, each message received by the controller corresponds to one event which triggers the transition of the state machines. Specifically, there are 6 messages received by the controller which corresponds to 6 state transitions in the state machine. Also, the controller forwards messages to relevant sensors or the gateway after receiving these messages, which corresponds to the 6 actions of the transitions. Therefore, the state machine is consistent with the controller lifeline in the sequence diagram.

Question IV

Figure 8 is sufficient to model the behavior of the controller. However, there are also situations that are not handled by this state machine which could lead to errors or deadlock during the runtime of the controller. For example, if the sensors never return observed value to the controller, the controller would get stuck in waiting states forever. Therefore, it is better to return back to Idle state after some time waiting for the sensor values. Also, the controller should also handle the situation in which sensors return incorrect values (e.g. requesting for temperature value but receiving humidity values). Figure 9 illustrates the enhanced state machine which represents those aforementioned situations.

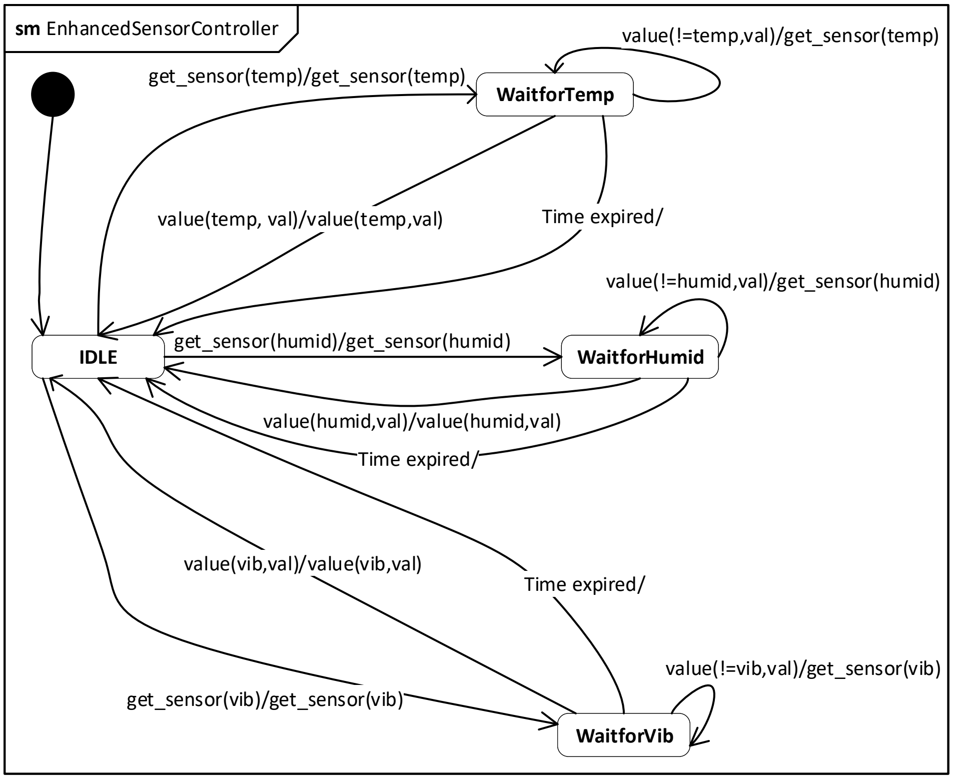

sm EnhancedSensorController
value(!=temp,val)/get_sensor(temp)
get_sensor(temp)/get_sensor(temp)
WaitforTemp
value(temp, val)/value(temp,val)
Time expired/
value(!=humid,val)/get_sensor(humid)
IDLE
get_sensor(humid)/get_sensor(humid)
WaitforHumid
value(humid,val)/value(humid,val)
Time expired/
value(vib,val)/value(vib,val)
Time expired/
value(!=vib,val)/get_sensor(vib)
get_sensor(vib)/get_sensor(vib)
WaitforVib

Figure 9: Enhanced state machine of the Sensor Controller