Q1 (Part 2)
The LTL property is:
G(WCPstatus == enable && clientsStatus[x] == idle -> F(clientStatus[x] == postupd))
where x are all integers from 0 to NUM_CLIENTS – 1 (inclusive of 0 and NUM_CLIENTS – 1)

This means that whenever the WCP is enabled and a particular client is connected and idle, that particular client will eventually go into the post-updating state (which effectively means that it has successfully gotten the new weather that was generated by the weather update i.e. Step 4 of the requirements under Weather Update).

Q1 (Part 3 and 4)
Deadlock:
(A screenshot of the counter example cannot be obtained as the message sequence chart is too large to fit in 1 screenshot, the isf file which has the message sequence chart can be found inside the Question1 folder and can be restored to see the message sequence chart.)

Description of counter-example from SPIN:
The deadlocked state that I have obtained is: WCP status == disabled, CM status == initializing, client0 status == idle, client1 status == initializing, client2 status == idle

This deadlock happens because of such a sequence:
1) There are 2 clients connected to CM and in idle state.

2) The last client sends connection request to CM, and proceeds as per normal until step 3 of requirements under Client Initialization (so the CM tells this last client to get new weather and sets the last client's status to initializing).

3) When the last client reports success for getting the new weather, the CM receives this report for success of getting the new weather and thinks that this report for success of getting the new weather is part of Step 4 of requirements under Weather Update, and not Step 4 of requirements under Client Initialization, because there is no check of the status of CM before executing Step 4 of requirements under Weather Update.

Ideally, there should be a "When CM's status is updating" in front of Step 4 of requirements under Weather Update, just like how there is a "When CM's status is post-updating" and "When CM's status is post-reverting" in front of Steps 5 and 6 of requirements under Weather Update.
As there is no such check, when CM receives a report for success of getting the new weather from any client, it can proceed to execute Step 4 of requirements under either Client Initialization or Weather Update. In this case, it was meant to execute Client Initialization, but went to execute Weather Update instead.

4) Once the CM has received this report for success of getting the new weather from the last client, it waits for a report for success/failure of getting the new weather from the other 2 idle clients, as it thinks that it is executing Weather Update. However, since the other 2 idle clients never received an instruction to get new weather, they will not send a report for success/failure of getting new weather. Thus, the program is deadlocked as the CM waits infinitely for a report which will never come, and all clients are not in a disconnected state (so they can't send a connection request to CM), and WCP is disabled (so it can't send a update request to CM).

For the verification trail sequence given in the isf file or the trail file, the error starts at line 9980, where when client1 tries to connect and sends a report for success for getting weather as part of Client Initialization, the CM interprets it as a report for success for getting weather as part of Weather Update, thus it increases the counter numClientsReportSuccGetWeather, and waits for the idle client0 and client2 to report back success/failure for getting weather, which will never arrive since client0 and client2 never received any instruction from CM to get weather.

Solution to make model deadlock-free:
To make the model deadlock free, there should be a statement: "When CM's status is updating," at the front of Step 4 of Weather Update.

If this is added, there will be no mix-up between Client Initialization and Weather Update, and only clients who have successfully completed Client Initialization can do Weather Update.

Thus, the deadlocked state which requires WCP status to be disabled and all clients to be connected (i.e. not disconnected status) will never occur, because when Client Initialization is being done, only 3 scenarios can happen:

1) Client reports success for getting weather and using weather, which results in client going to idle and WCP being enabled.
2) Client reports success for getting weather but failure for using weather, which results in client going to disconnected and WCP being enabled.
3) Client reports failure for getting weather, which results in client getting disconnected.

And all 3 scenarios will not lead to the deadlocked state which requires WCP status to be disabled and all clients to be connected.

Additional problems:

1) **Deadlock:** In situations where all clients are required to report success or failure, for example in Step 5 and 6 of the requirements under Weather Update, and if there are multiple clients connected, there are 2 different ways of implementing a situation where a client reports failure but there are still clients who have not reported back success/failure.

With reference to Step 5 of the requirements under Weather Update, if 1 client reports back failure for using new weather, the 1st implementation is for CM to wait for all the other clients to report back success/failure for using the new weather, before disconnecting all clients, re-enabling WCP, and setting its own status back to idle (because one of the clients reported failure for using the new weather).

The 2nd implementation is for the CM to straightaway disconnect all clients, re-enable WCP and set its own status back to idle, without waiting for the other clients to report back success/failure.

The 2nd implementation could cause a deadlock, because the other clients' reports for success or failure for using new weather will only come in after CM sets its own status back to idle. As the requirements do not state that the CM should be able to understand a "success/failure for using new weather" message while it is in idle status, the CM will receive this message but not be able to interpret it, thus causing a deadlock.

Thus, the deadlock arises because of an implementation bug, whereby the CM has received a message that it is not supposed to receive while in idle status.

Solution to make model deadlock-free: I have implemented according to the 2nd implementation, but added an "else -> skip" in line 288, so that if the CM receives a message from a client that it cannot interpret, it will simply skip and discard the message and continue on executing a next statement, and will not get deadlocked there.
Alternatively, implementing using the 1st implementation will also cause this deadlock to not occur, as the CM will wait for all messages it is supposed to receive in that state to come in first, before moving on to another state.

2) **Deadlock:** Under Step 2 of the requirements under Weather Update, it is stated what should happen "When CM is idle and receives update request from the WCP, …". However, it is not stated what should be the expected behaviour of the system if CM is not idle and receives update request from the WCP. This is not like Step 2 of the requirements under Client Initialization, where it is stated what should happen "if CM's status is idle when connecting request is received", and what should happen "otherwise, if CM's status is not idle (when connecting request is received)."

Thus, if CM is not idle and receives update request from the WCP, the CM will receive the message but not be able to interpret it, thus causing a deadlock.

Solution to make model deadlock-free: I have added an "else -> skip" in line 309, so that if the CM receives a message from WCP that it cannot interpret, it will simply skip and discard the message and continue on executing a next statement, and will not get deadlocked there.

3) **Error with protocol:** Because of the problem mentioned in the 1st deadlock, a client that is in client initialization and sends a report for failure to get weather could also cause the CM to send messages to all clients to use old weather information and set its own status and the clients' status to post-reverting (as per Step 4 of Weather Update), instead of disconnecting the client that is undergoing client initialization and setting its own status back to idle (as per Step 4 of Client Initialization). This is not a desirable outcome as the client should be ideally disconnected.

4) **Livelock:** There is a possible trace whereby one or all of the clients keeps trying to initialize/connect, but when CM requests it to get new weather, it keeps reporting back that

it failed to get new weather, thus resulting in it getting disconnected, and this keeps repeating infinitely.

Although this is not a deadlock, as there is always an outgoing edge from a state where the client is disconnected (it can keep trying to request connection again), this represents a possible livelock situation, as there can be an infinite loop of the client requesting connection and getting disconnected.

Also, in Step 4 of the requirements under Client Initialization, if getting new weather fails, the CM only disconnects the client and sets its own status back to idle but does not re-enable WCP. Thus, if the infinite loop of the client requesting connection and getting disconnected because it reports failure to get weather occurs, WCP will never be able to get enabled, and this is not a desirable outcome, as it means that system users will never be able to manually update new weather information, as it is stated in Step 1 of the requirement under Weather Update that “users can manually update new weather information only when the WCP is enabled”.

Q2 (Part 2)
The sequence of events is as shown in the isf file. There are 3 shuttles: shuttle0, shuttle1, and shuttle2 (which correspond to s_1, s_2 and s_3 in the requirements respectively).

1) At the start, all shuttles are initialised, and all tracks are initialised to be available.
2) When the 1st order is sent out to all shuttles, shuttle2 replies first with acceptance of order, because it fulfils the criteria of “current loaded size plus order size does not exceed capacity” and “start destination of order is within 2 stations away from its current position”. Since at this point, only shuttle2 has replied, the lowestOffer is set to shuttle2’s charge of $3, and shuttleToReply is set to shuttle2.
3) Shuttle1 replies next with acceptance of order, but since its charge is $4 which is higher than the lowestOffer, lowestOffer is still set to shuttle2’s charge of $3 and shuttleToOffer is still shuttle2.
4) Shuttle0 replies next with acceptance of order, and since its charge is $2 which is lower than the lowestOffer, lowestOffer is set to shuttle0’s charge of $2 and shuttleToOffer is now set to shuttle0.
5) Management system checks that all shuttles have replied, and sends shuttle0 a message that the order has been assigned to it, and shuttle0 receives this message and has the order assigned to it.
6) Since shuttle0 is not transporting any orders, and since it is at station1 which is the start station of the 1st order, it picks up the order and its currLoad is now 4. It calculates that the nearest way to station3 (destination station of the 1st order) is to go to station2, so it goes onto the track going from station1 to station2, and then from there goes to station2.
7) At this point, shuttle0 is at station2, shuttle1 is at station1, shuttle2 is at station2.
8) Management system now sends out the 2nd order to all shuttles. The procedure is as before, with shuttle2 replying first, then shuttle1, then shuttle0 (all reply accept offer), and shuttle0 getting the assignment at the end as it offers the lowest charge.
9) Shuttle0 now moves onto the track from station2 to station3, and then goes to station3. (It does not pick up the 2nd order at station2 because loading or unloading at other stations is

not permitted while it is completing/transporting an order, and currently it is transporting the 1st order.)
10) Shuttle0 drops off the 1st order at station3, then moves onto the track from station3 to station2, and then goes to station2. Here, it picks up the 2nd order from station2.
11) Shuttle0 then moves onto the track from station2 to station3, and then goes to station3, and drops off the 2nd order at station3.
12) All shuttles are now at stations without load. (Shuttle0 is at station3, shuttle1 is at station1, shuttle2 is at station2.)

Each shuttle has an array of ordersBeingExecuted and ordersAssigned, whereby ordersBeingExecuted are orders that it is currently transporting, and ordersAssigned are orders that it has been assigned by the management system but has not picked up yet.
For movement, each shuttle will take the nearest way to go to a station if it has to go to that station to load/unload an order. If it is not transporting any orders and has no orders assigned to it, the shuttle will stay put at the station.