Australian National University

ASSIGNMENT COVER SHEET

ANU College of Engineering and Computer Science
Australian National University
Canberra ACT 0200 Australia
www.anu.edu.au

+61 2 6125 5254

Submission and assessment is anonymous where appropriate and possible. Please do not write your name on this coversheet.

This coversheet must be attached to the front of your assessment when submitted in hard copy. If you have elected to submit in hard copy rather than Turnitin, you must provide copies of all references included in the assessment item.

All assessment items submitted in hard copy are due at 5pm unless otherwise specified in the course outline.

Student ID	U6492211		
For group assignments, list each student's ID			
Course Code	COMP2310		
Course Name	Systems Networks and Concurrency		
Assignment number	1		
Assignment Topic	Swarm		
Lecturer	Uwe R. Zimmer		
Tutor	Michael Bennett		
Tutorial (day and time)	Monday 9:00		
Word count		Due Date	28/09/2018
Date Submitted		Extension Granted	

I declare that this work:

☑upholds the principles of academic integrity, as defined in the ANU Policy: Code of Practice for Student Academic Integrity;

☑is original, except where collaboration (for example group work) has been authorised in writing by the course convener in the course outline and/or Wattle site;

☑is produced for the purposes of this assessment task and has not been submitted for assessment in any other context, except where authorised in writing by the course convener;

☑gives appropriate acknowledgement of the ideas, scholarship and intellectual property of others insofar as these have been used;

☑in no part involves copying, cheating, collusion, fabrication, plagiarism or recycling.

Initials

For group assignments, each student must initial.

Sukai Huang

COMP2310 Assignment 1 Report

Brief Introduction about the work

The current system fulfills all the requirements from stage a to d. The system allows the swarm group to reach a consensus that who will become the leader, and then the leader will decide who will survive and also schedules the "feeding" activity so as to avoid traffic congestion. The system also provides solutions for the unexpected situations. For example, the system allows the swarm group to vote for a new leader when the previous leader swarm breaks down, or, allows the swarms to "rescue" themselves when their charge is low. It thus enhances the sustainability and maintainability of the system.

All in all, the basic idea behind this mechanism is that it allows the swarm groups to reach a consensus and then build up an authority (swarm leader) to schedule their behaviors under a distributed environment. Below is the further explanation.

The main problems in this assignment

1. Consensus

It is shown that traffic congestion happens if all the swarms want to reach the globe at the same time. Therefore, the feeding activity must be scheduled. Moreover, the swarm group will have to decide who will survive in stage d. It seems that this assignment requires the swarms to reach a consensus on many of things. However, the issue is that which swarm should they listen to? Or, if a swarm received an 'order' message, should it follow?

2. Message passing under distributed environment

Since there is no shared message database, the swarms must pass their message as well as receive other's message. Under distributed environment. the latest message one receives may be outdated already. Below is a simple example call "I see miracle":

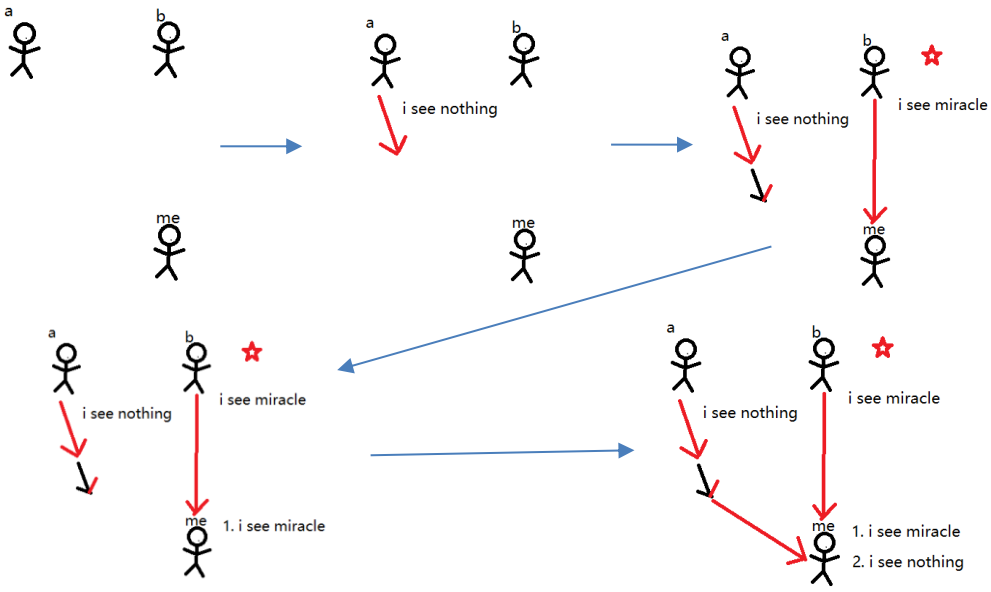

Figure 1: I see miracle example

Therefore, the system needs to find a solution to deal with this issue so that the swarm will only synchronize the correct message rather than just overwrites the local database when new message comes.

Design

The design is continually upgraded during the period. The latest Design Model is presented below

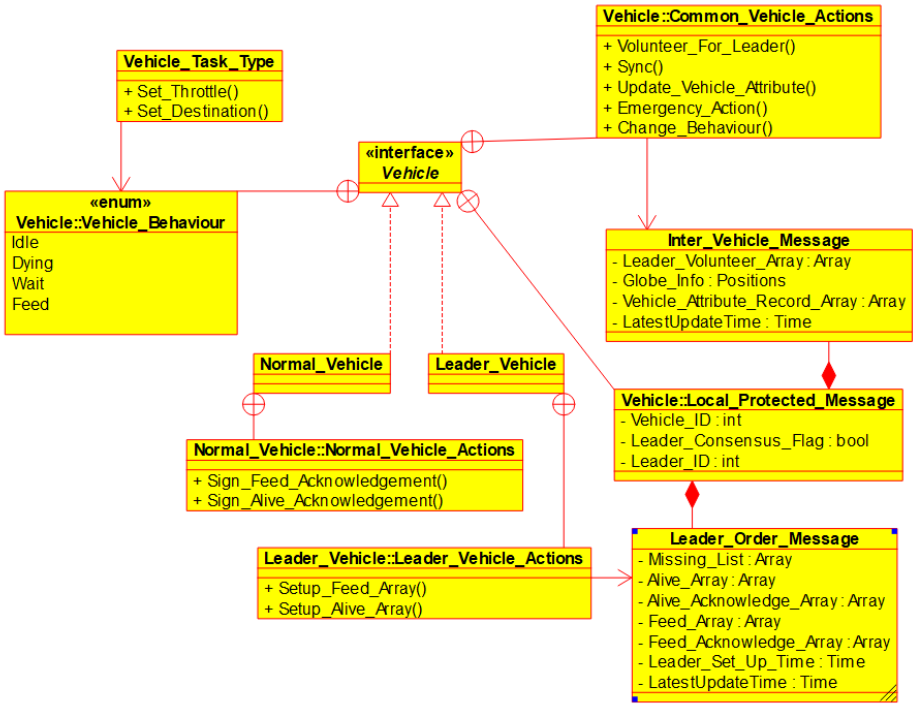

Figure 2: Design Model UML

The main flow of this design is that, first of all, the swarm group will decide a leader. (The decision-making process will be discussed in the Implementation section). Then the leader vehicle will come up with a schedule and send out this schedule message (Leader_Order_Message). The follower vehicles will then follow the leader's schedule and then change their current 'mode'. This design also requires the 'selected' vehicles (e.g. selected for 'getting feed') to send acknowledgement message back to the leader so that the leader can modify its schedules according to the acknowledgement feedback. Moreover, since the leader is essential in this system, the leader must send signals constantly to prove that he is still working. The followers will vote for a new leader if they could not receive this 'verification signal' within a certain period. These mechanisms help to enhance the sustainability and maintainability of the system.

The vehicles will set their throttle and destination based on their current 'behavior/mode'. This model defines 4 types of vehicle behaviors. "Idle" is the initial mode of the vehicles, and the 'idle' vehicles will all move to the globe and then decide their leader. The vehicles that are on "Feed" mode will move to the globe too to get recharge. Vehicles that are on 'Wait' mode will prevent themselves from getting too close to the globe, so that the "Feed" mode vehicles will not be blocked. The vehicles under 'Wait' mode also have emergency-handling mechanism, which allows them to move to the globe regardless of the schedules to avoid dying. The mode of the vehicles is decided by the leader.

The 'Leader-Follower' Design solve the consensus problem I claimed in the previous section. However, there could be "Leaderless" design for this assignment. Nevertheless, the 'Leader-Follower' model is considered to be more robust and relatively simpler to build compared to the 'Leaderless' structure. Having a leader seems like a 'remote central coordinator' in this circumstance.

The database model is also essential in this design. The type of the data is array type. The vehicle will store its information into one of the blocks of the array based on its ID and backup other vehicles' information in other blocks. The database also stores globe information and the leader's scheduler information.

Implementation and Analysis

There are lots of mechanisms in this design. Below are the some of the key implementations that makes this design unique and interesting.

1. Message synchronization

To avoid the message passing problem I claimed above, all the information is associated with an “update time”. As such, by comparing the incoming data’s update time with that of the local data, each vehicle will ensure that they will only store the latest information into his database. It is important that different data cannot share the update time. I made this mistake in the previous design and it caused data missing.

2. Consensus on leadership

Reaching a consensus on leadership is the main part of this model. A trivial solution could be just assigning a particular one to be the leader (e.g. No.1 swarm becomes leader). In fact, this can completely work, but it is not fair and it is not a ‘consensus’ in the normal sense either. As such, I was thinking again and again, and then came up with this solution by myself. (I can proudly say that I did not search any references or articles) I would like to write down the details of my thinking flow. So, I was thinking of the term “mutual exclusion” (See figure 3) Since only one can be the leader, we hence could ask the vehicles to check who has volunteered, if no one volunteers (ldrex), then this vehicle volunteers and then tell the public to get consensus. However, it is possible that before this vehicle tells the public, some one may volunteer and tell the public during this time interval, thus this vehicle needs to check again before he finally publish his ‘volunteering’ declaration (strex). Also we need to consider that it requires time for message passing. Thus we wait for a certain time (e.g. 2 seconds) and then if there is no new declaration coming within this period, then we can safely claim that we reached a consensus.

```
ldrex r0, [r3]
cbnz  r0, fail_enter  ; if locked
mov   r0, #1          ; lock value
strex r0, [r3]        ; try lock
cbnz  r0, fail_enter  ; if touched
dmb                   ; sync memory
```

Figure 3 Mutual Exclusion

Another way of choosing the leader could be that vehicles claim to be the leader and provide the time when they claim. And then after a certain time the vehicles analyze all the volunteering declarations they received and vote the first one who claimed it. In this case we need to assume that the ‘clock’ is universal across the system.

3. Scheduler making with Acknowledgment

The leader will decide who will live and also schedule the ‘feeding’ activity. The key point is that it is more sustainable and maintainable to have acknowledgement mechanism. This mechanism allows the leader to know that the target vehicles in the schedule list is missing or not. It is especially important when deciding who will survive in stage d, where the leader needs to make sure that just 42 vehicles will stay. If the design does not have this acknowledgement mechanism, then it is possible that the leader would put a missing vehicle into the “Staying” list.

4. Prevent traffic congestion

The vehicles that are under ‘Wait’ mode should stay away from the globe. But the question is how far away should the vehicle stay.

Initially I was thinking that vehicles store their relative positions to the globe when they first reach consensus on leadership. And after that they stay there when they wait. However, I found out that this solution could not solve traffic congestion efficiently. My tests clearly show that it is better for vehicles to constantly move rather than stay at an appointed position. (treat globe as reference position)

The current solution is to let these vehicles move in and out when they wait for instructions. The advantage of this solution is that these vehicles may find isolated vehicles and rescue them.

5. Leader missing situation

This model is designed for only one leader. However, we must also solve the situation when the selected leader is missing. In the circumstance of two or more globes, it is possible that some of the vehicles go to different galaxy and lose contact with the current leader. My solution is that when the vehicle lose connection with its leader, it will become a leader and lead itself.

If everyone is leader then there will be no regulations anymore, so my next rule is that when a leader receives another leader’s message (It also means that this leader is not isolated), the leader will compare the leader declaration time. If the other leader declared his leadership earlier, then the young leader will give up his leadership and then follow the senior leader. In this case we can decrease the number of useless leaders and control the vehicles in a good regulation.

Potential problems and Improvements

1. Information integrity

The current code does ensure information integrity to some extent. The procedures that modify the leader's scheduler message will check whether the caller is a leader. If not, it will raise exception. However, it is just a trivial information integrity solution. Vehicles would not know if the sender modify or fake this passing information. In the future we may need a professional information integrity technique. (e.g. having witness mechanism)

2. Code structure

It would be better to separate some code into different packages so as to make the code architecture more concise. Also, some condition checking in the code is redundant. Due to the time constraints I am not able to improve this. (Code reconstruction often causes many bugs) In the future, I may create a design model UML first and then construct the code according to it so as to make the architecture clearer and better.

3. No further substitution mechanism

After the first leader finished deciding who would stay, the 'Survival List' will no longer change. The current design still lacks the mechanism to support the case where a new vehicle (using '+' to add new vehicle) can become a substitute if some local vehicles accidentally died. Nevertheless, this case rarely happens because there are algorithms in the design that help to prevent vehicles from using up their charges.

For further information, one can check the code comments in the source code file. One can also observe the terminal text output to get more information of this system in run time.