Drone Simulator in a smart city

Group 2

This document describes in a global way the understanding that we had of this project of drone simulator in a smart city.

Summary

1. Introduction

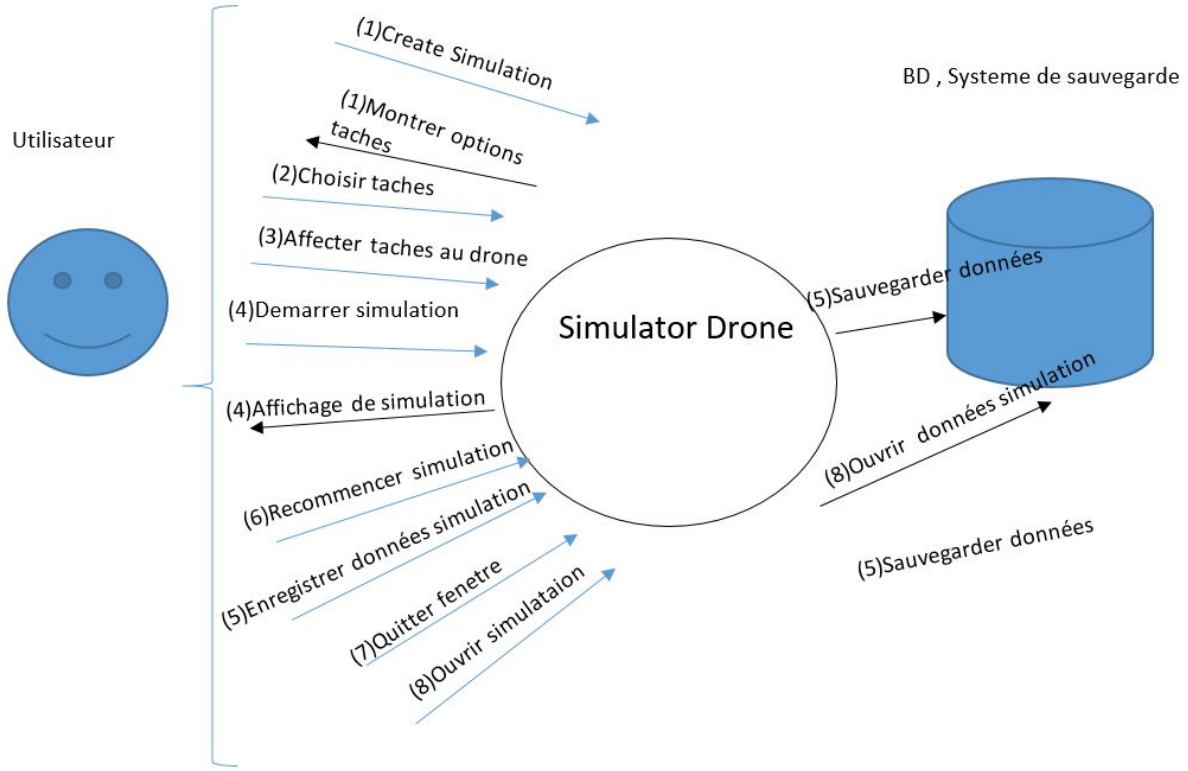
As part of the GL52 project our goal is to implement a multi drone agent simulator. There are: surveillance drones and delivery drones. Surveillance drones have the mission to explore their environment in order to find a intruder. While delivery drones have the mission to deliver packages in buildings avoiding obstacles in their path.

To realize this project, at first we define the environment as a city with buildings and trees. Then we will put the different drones in the environment. Finally in a last part we will detail the behavior of drones.

2 - General description

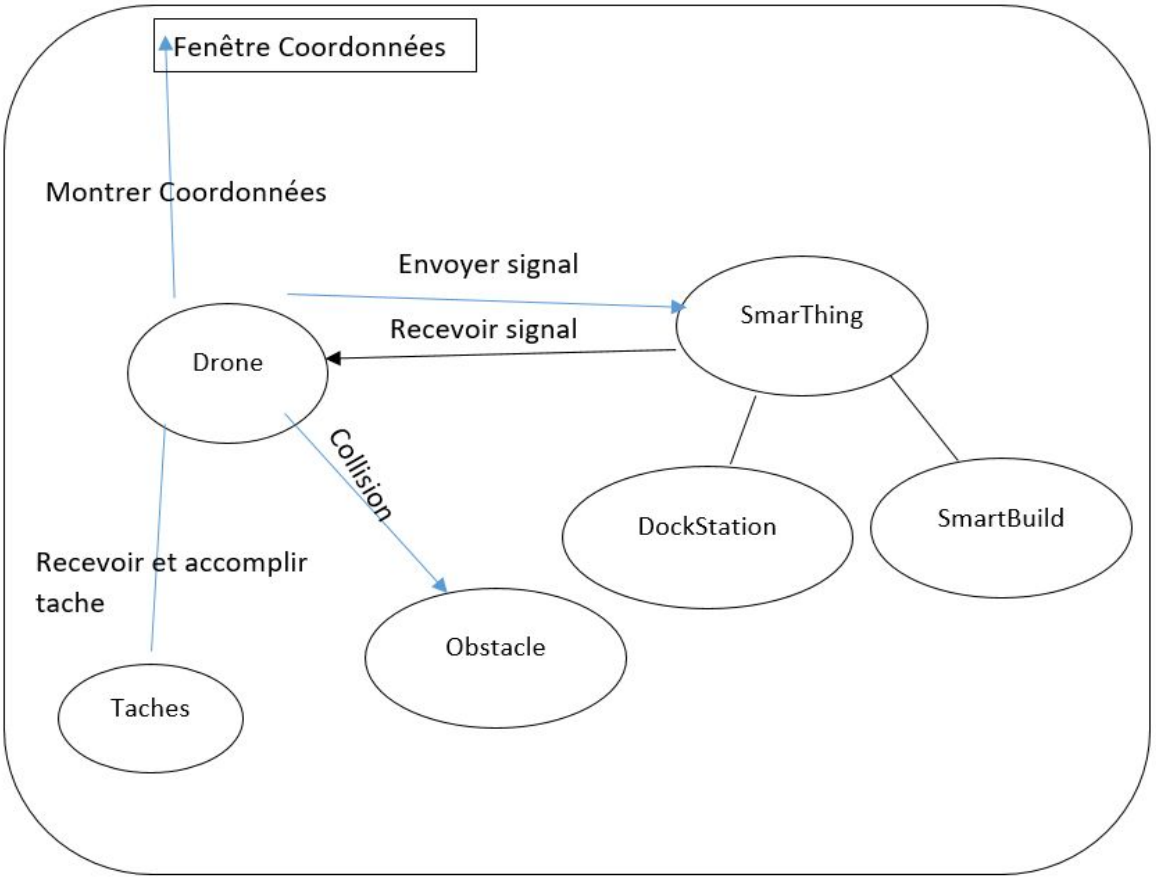
Context of system

Context diagram of User requirements



Functionalities	***Description***	***Secondary Actor***
Choose tasks	*- List of present tasks where the user can choose one or more tasks to assign to the drones*	
Assign tasks to a drone	*-Allows the user to assign one or more tasks to one or more drones before the start of the simulation*	- *Drone*
Start a simulation	*-Allows the user to start a new simulation or to return to an already existing simulation*	
Restart a simulation	*-Start a new simulation with the same characteristics as the current simulation*	
Save data	*-Allows the user to save the current simulation data*	
Quit a simulation	*-Allows the user to end a simulation*	

Detailed diagram Drone Simulator(Low Level View)



Some features are not shown in the diagram above because they are nested in one of the features described in the table below.

Functionalities	***Description***	***Secondary Actor***
Start a task	*-The drone will position itself at the desired place* *- He will start the task triggered by the user*	
Do tasks	*-The drone will perform the tasks that the user assigned to him, such as joining a dock Station, reach a point B ..*	*- Dock Station* *- Smarting building* *- Obstacles*
Communicate (Send / Receive signal)	*-Communicating with drones to know their positions* *-Contact the Dock Station to know the availability* *-Communicate with smartthings*	
Stop	*-It stops only if it has completed its task(s) or if the user has to stop the simulation*	*- USER*
Collide	*-A drone can collide with non-smart obstacles*	*- Obstacles*

User's characteristics

First there is the user: USER, who can assign tasks to a drone, start a simulation session, restart a simulation, save data or even quit a simulation.

Then there is a user as an autonomous agent (drone), it will perform tasks (join a point B, join a dock station ...), it will communicate with other drones, dock stations, connected objects (smart building , smart things ...), the simulation is finished when the drone has finished all the tasks that have been assigned.

And in the system, there are the dock stations: who will send the drone its availability or not and smartthings: who will send data to the drone.

Development constraints

We decided to develop this project in Agile mode by an incremental model because the project is complex in nature, so it will be necessary to prioritize certain features.
We will use an object-oriented development language (JAVA) and UML as the modeling language.

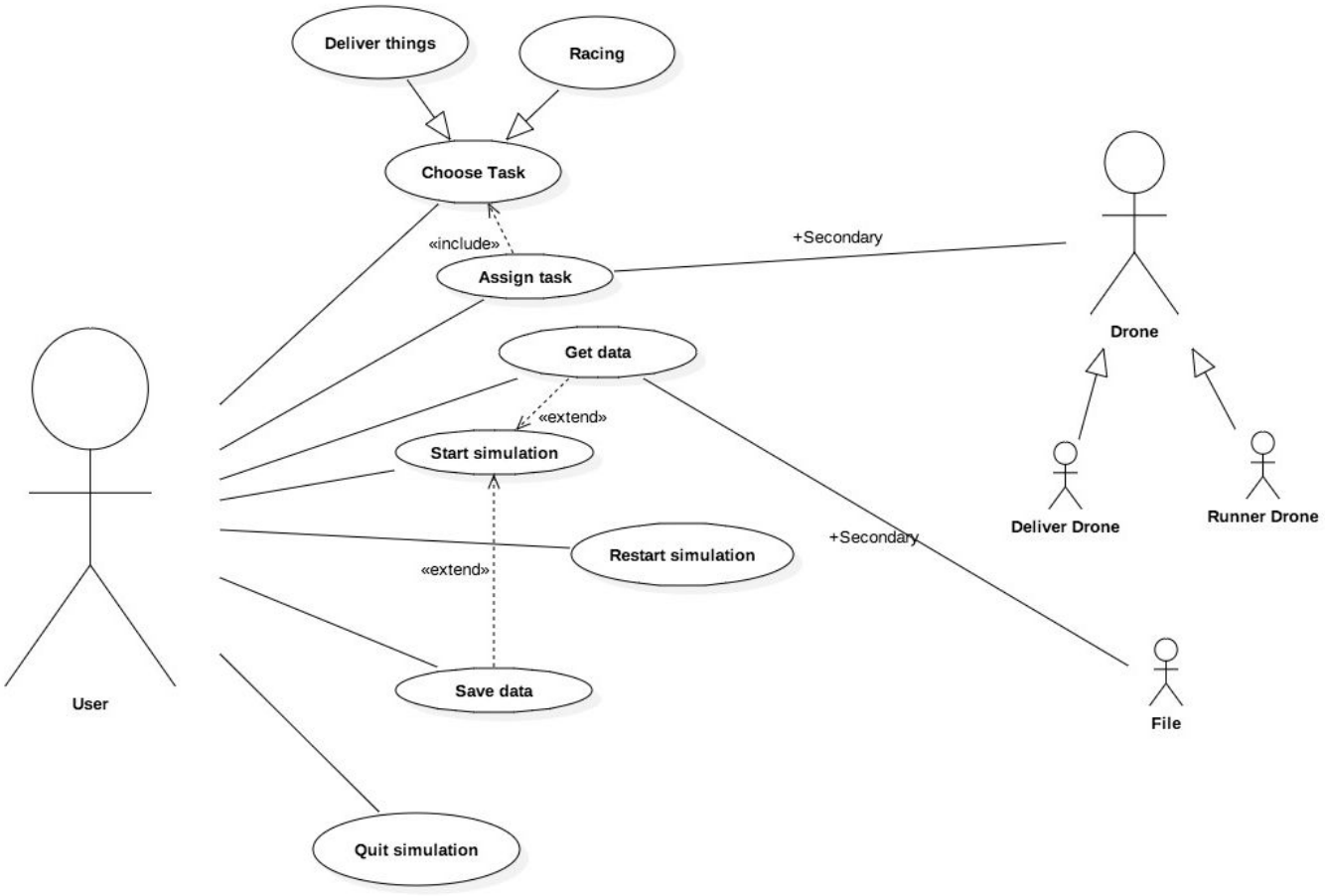
In a first instance the project will be implemented as a desktop application and we will prioritize English as a communication language on our source code.

Working hypotheses

Factors likely to call into question all or part of the realization of specifications :

- Mastery of the JAVA language
- Development time
- Understanding / lack of knowledge of multi-agent systems
- Lack of resources (developers,hardware)

Use Case - User



Name	***Description***	***Primary Actor***	***Secondary Actor***	***Pre-condition***	***Post-condition***
Choose task	*-Two tasks can be chosen drones: deliver things or raceing*	*-User*	*-Drone*	*-Having a list of task : deliver things or race*	*-One task is placed on the scene*
Assign tasks to a drone	*-Allows the user to assign tasks according to the kind of drones*	*-User*	*-Drone*	*-The task is chosen*	*-The task has been assigned to the corresponding drone*
Start a simulation	*-Allows the user to start a new simulation or to return to an already existing simulation*	*-User*		*-Need a task*	*-The simulation is launched*
Restart a simulation	*-Start a new simulation with the same characteristics as the current simulation*	*-User*		*-Having one simulation running*	*- A new simulation is launched*
Save data	*-Allows the user to save the current simulation data*	*-User*	*-File*	*-Need a simulation data*	*-Data is saved in the file*
Quit a simulation	*-Allows the user to end a simulation*	*-User*		*-One simulation is launched*	*-A simulation is stopped*

3. Functional Requirements

1) User Use case

An user can do different things on this simulation. Documentation of what the user can do :

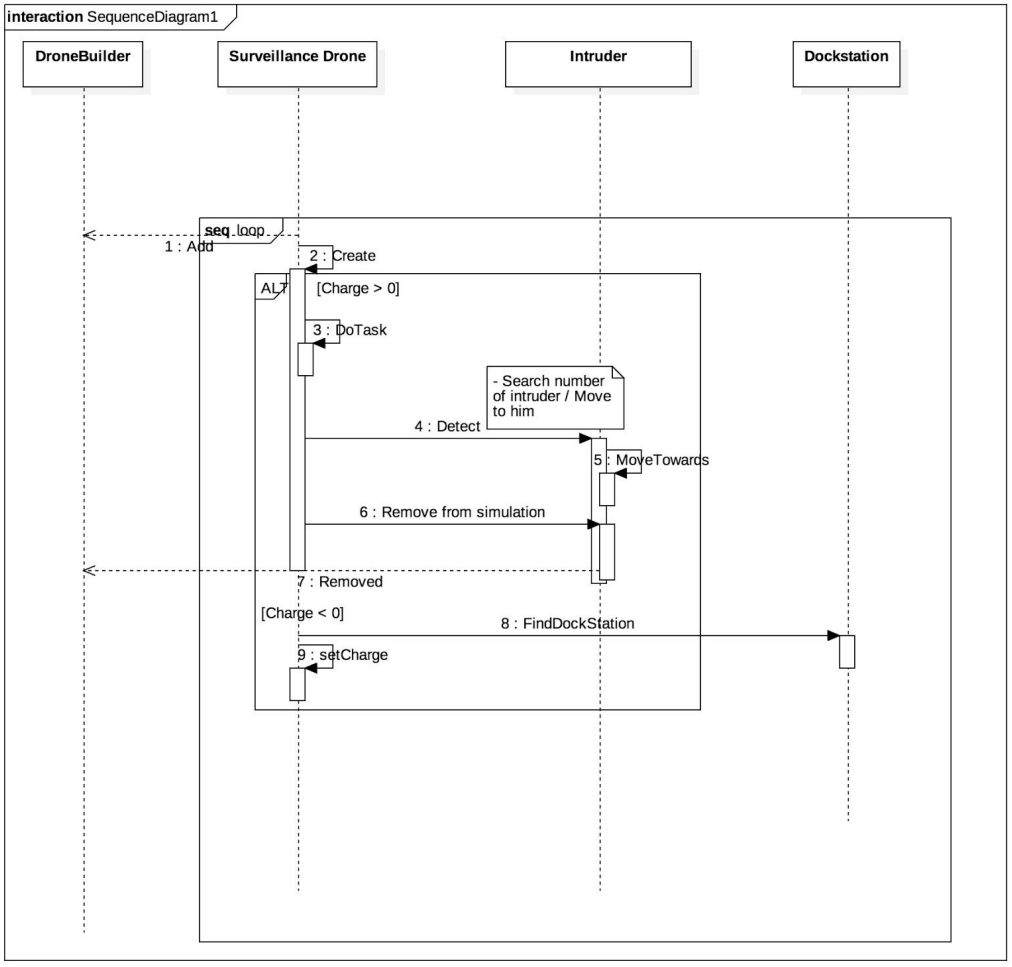
Name	***Description***	***Primary Actor***	***Secondary Actor***	***Pre-condition***	***Post-condition***
Choose task	*-Two tasks can be chosen drones: deliver things or oversight drone*	*-User*	*-Drone*	*-Having a list of task : deliver things or race*	*-One task is placed on the scene*
Assign tasks to a drone	*-Allows the user to assign tasks according to the kind of drones*	*-User*	*-Drone*	*-The task is chosen*	*-The task has been assigned to the corresponding drone*
Start a simulation	*-Allows the user to start a new*	*-User*		*-Need a task*	*-The simulation is*

	simulation or to return to an already existing simulation				*launched*
Restart a simulation	*-Start a new simulation with the same characteristics (parameters) as the current simulation*	*-User*		*-Having one simulation running*	*- A new simulation is launched*
Save data	*-Allows the user to save the current parameter of this simulation*	*-User*	*-Code of this simulation*	*-Need a simulation data*	*-Data is saved in the code*
Quit a simulation	*-Allows the user to end a simulation*	*-User*		*-One simulation is launched*	*-A simulation is stopped*

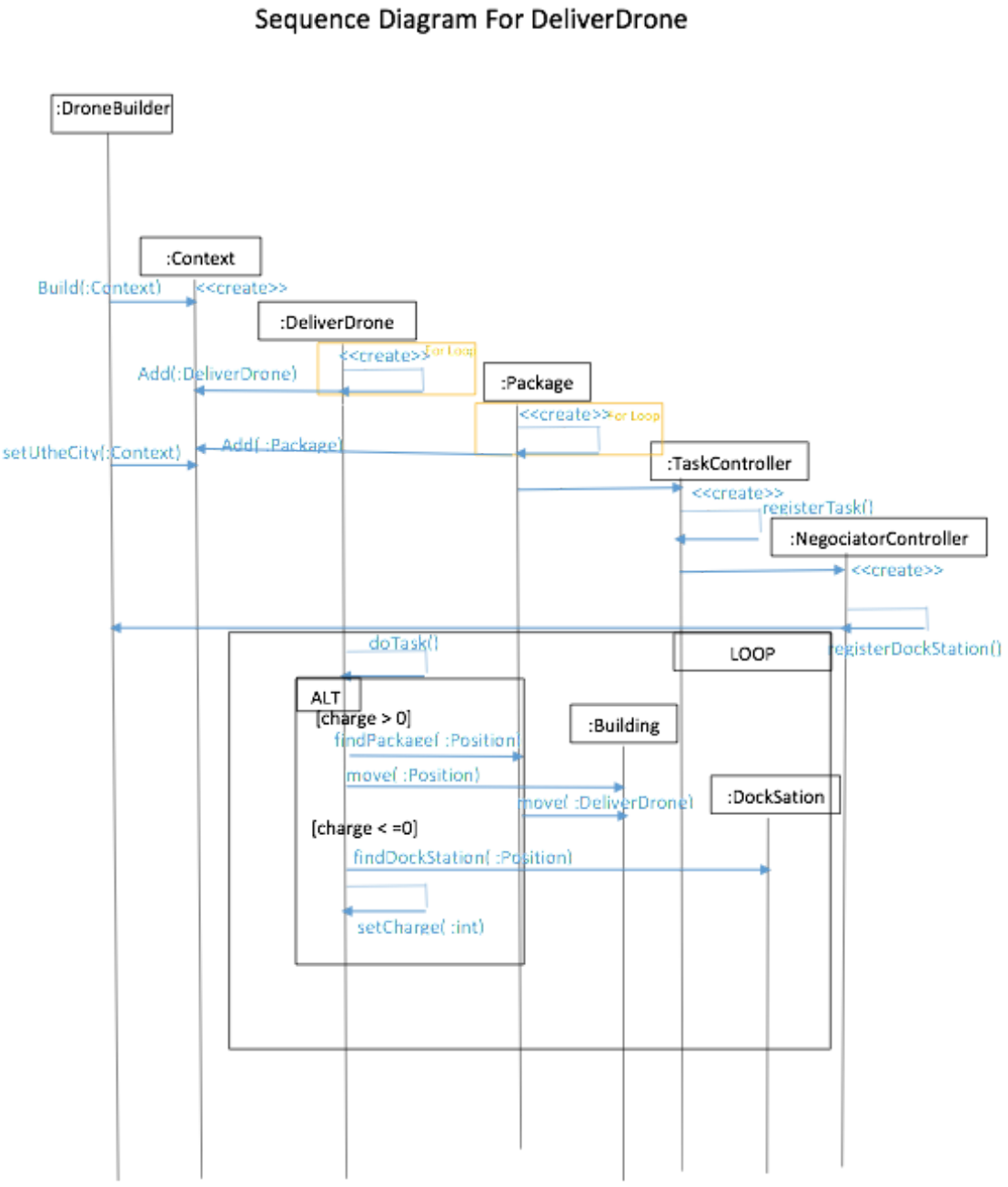
The parameters are the different number of drone (deliver, oversight…) that the user can be choose before to start the simulation.

2) Sequence diagram of the oversight drone (Surveillance drone)

The goal of the oversight drone (surveillance drone) is to find all the intruders in the simulation and to remove(kill) them to this simulation (so the intruder disappear when they are found by the oversight drone).

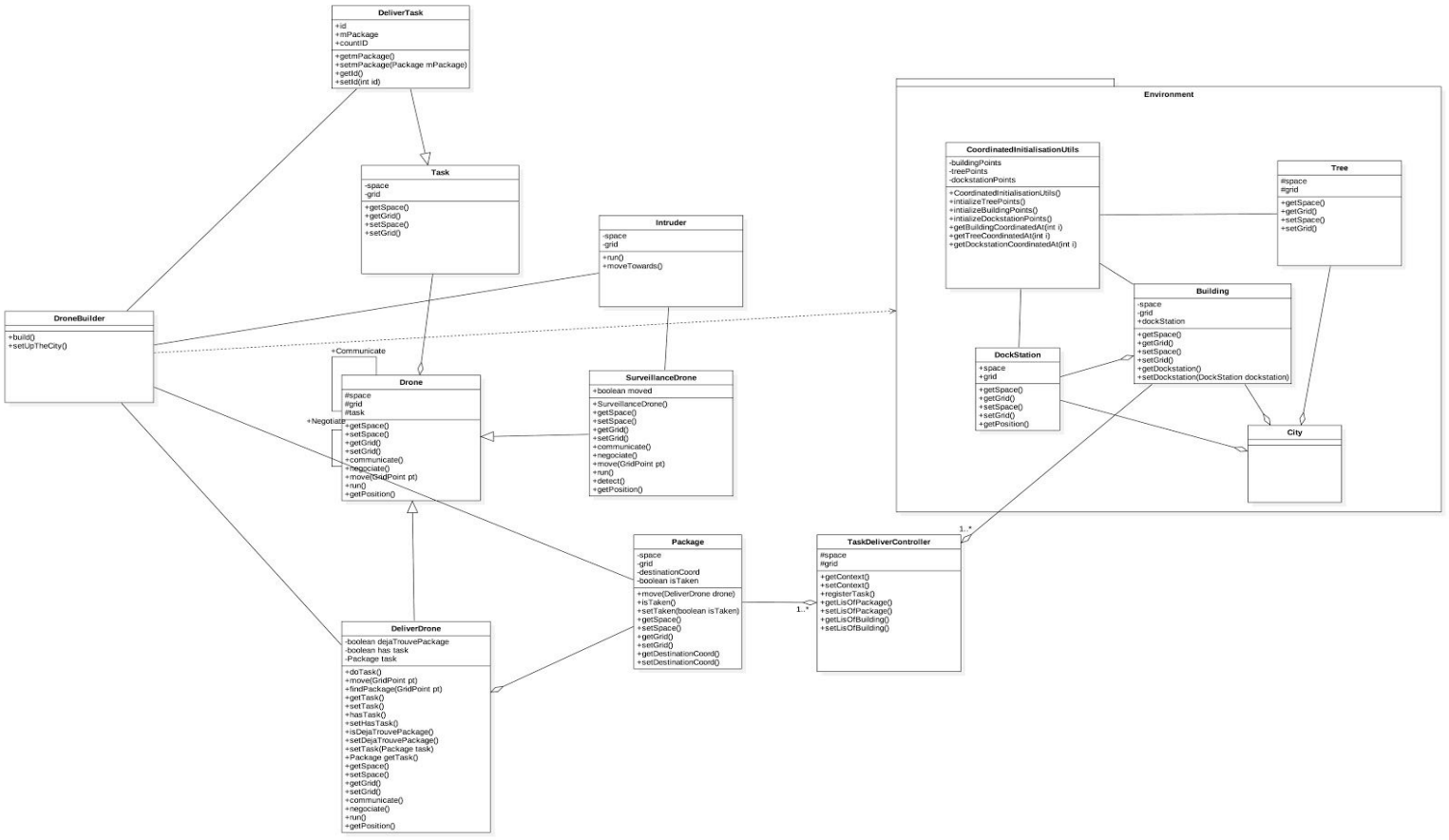


3) Sequence diagram of the deliver drone



4. Data structure’s Specification

There is a class diagram describing the entities of the system and their relations.



(This class diagram is in attachment of this project).

- A Tree is situated in a space and a grid.
- An Intruder is an agent situated in a space and a grid. It can move.
- A Package is an agent situated in a space and a grid. It have a destination coordinated and can be taken or not.
- A DroneBuilder is the context builder which allows the application to initialise the context and the city.

- CoordinatedInitilisationUtilis is an utility which provide the coordinated used by the builder to place the agents in the space.

- A Dockstation is situated in a space and a grid. It needs to be place on the top of a building.

- A Building is situated in a space and a grid, it can have a dock station or not.

- A Drone is an agent situated in a space and a grid. It can move and knows is own position.

- A SurveillanceDrone is a drone, which can detect an intruder and catch it.

- A Task is an object which represent a drone task.

- A DeliverTask is a task which have is own id and refers to a specific package.

- A DeliverDrone is a drone, which can find a package, and do a deliver task related to the package found.

5. Specification of external interfaces

1) Hardware Interface

Hardware configuration necessary for the operation of the application is :

- 124 Mo de RAM (Java 7) et 128 Mo de RAM (Java 8)
- 8GO de RAM or more

For more performance, it’s advisable to use a more powerful machine to avoid slowing down (which can happen with Repast sometimes)

2) Software Interface

For this project, it is necessary to have these different software :

- Java : 7 ou 8
- Eclipse : Neon or Oxygen, for the code (all classes are visible on eclipse and we can launch repast from Eclipse)
- Repast Simphony (modeling toolkit) : 2.5, it is a Java-based modeling system.

3) Software's User Interfaces

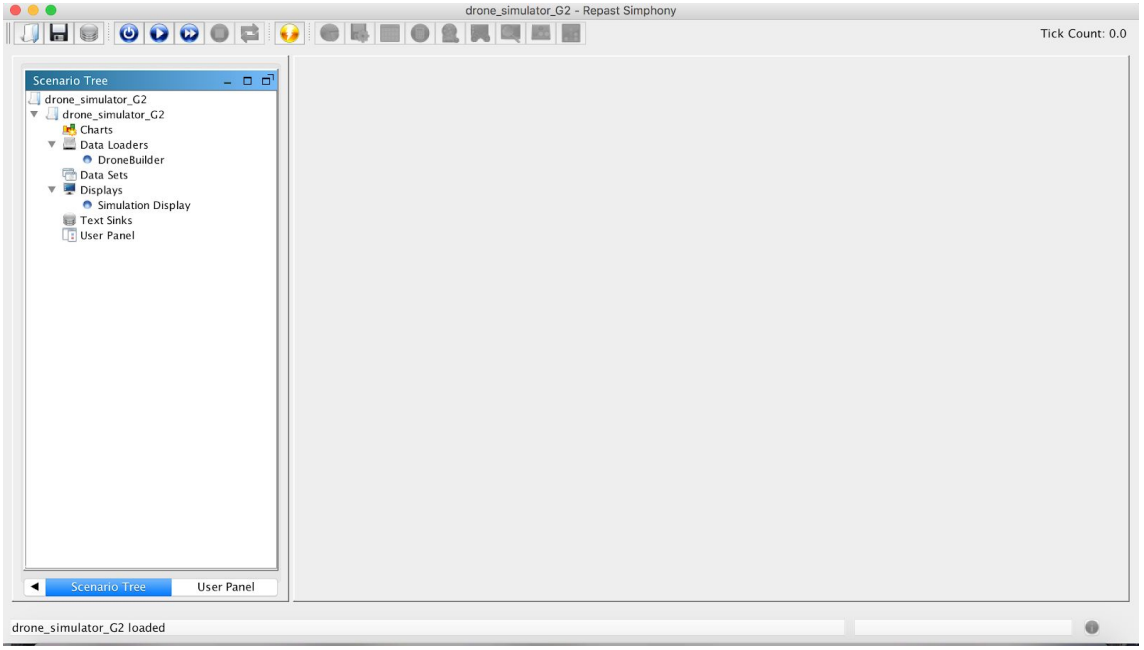


Figure 1 - Interface of the Application with no simulation

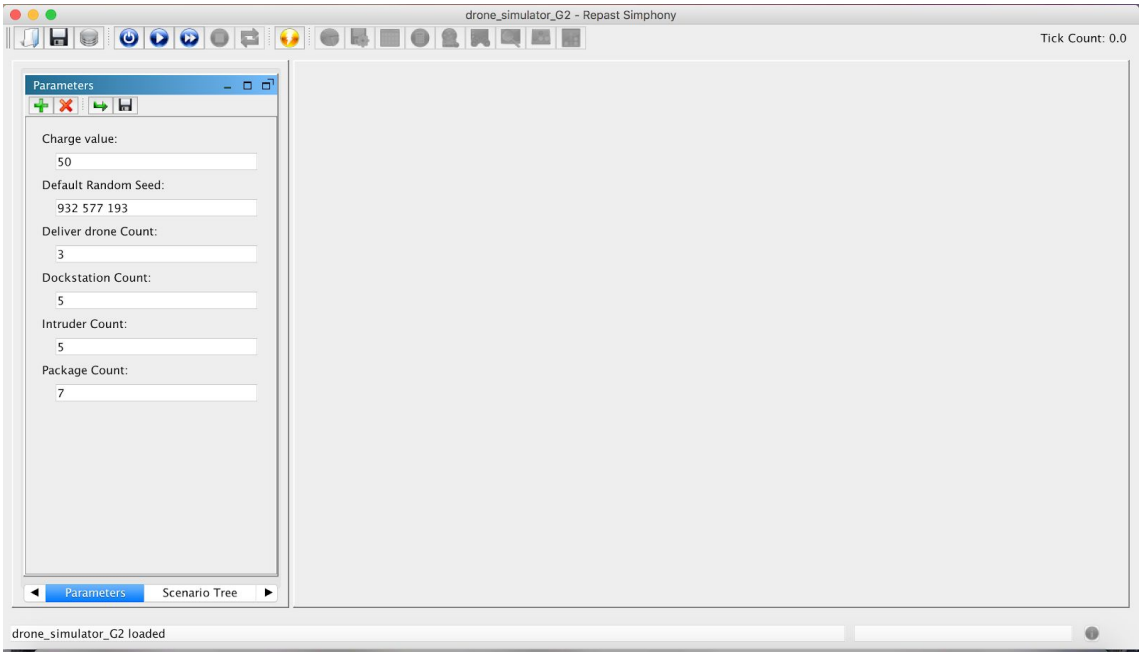


Figure 2 - Window of parameters of the application

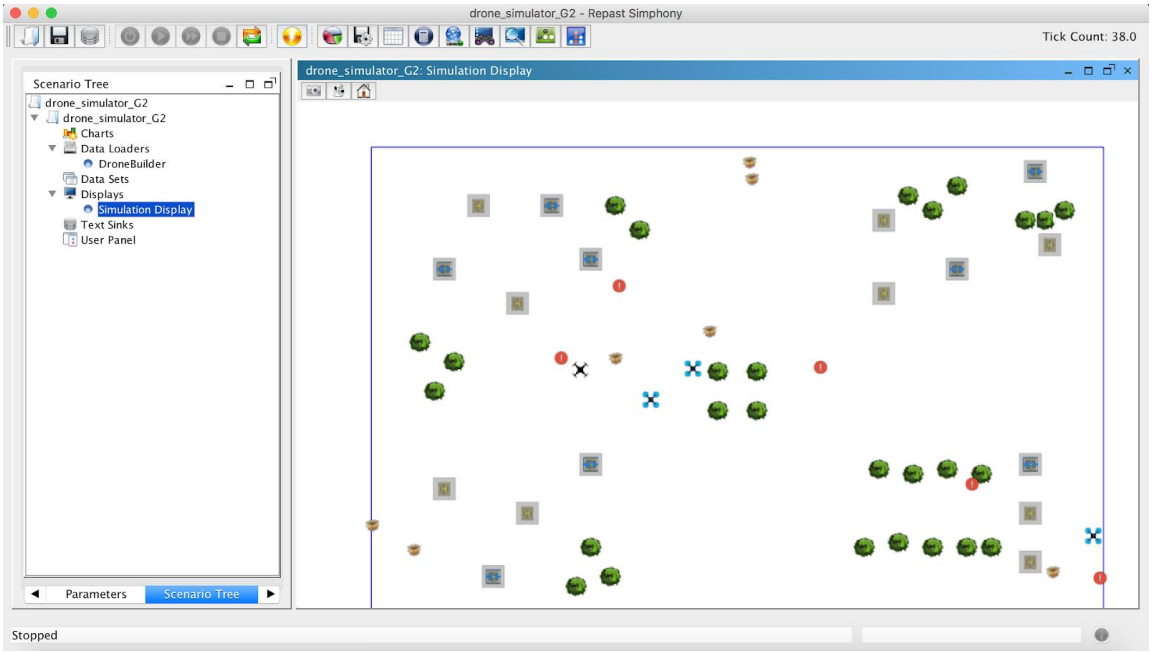


Figure 3 - Interface of the Application with simulation running

The User's interface (Repast interface use for this simulation) is composed of several parts :

- Buttons on the top bar (in Figure 1)(the most useful for the user are the play, pause, advance, stop buttons (when the simulation is running), and start again (when the simulation is running too).
- The left part consists of different things but the useful part for the user is the part "Parameters" (Figure 2) which is accessible by pressing the button to the left of scenario tree (Figure 1). In this part the user can put the number of deliver drone, intruder, package, the dock station and the charge value that he wants.
- The big part of right is the display of the simulation (Figure 3).

6. Performance Requirements

- To work properly, the simulator still needs to have a fairly powerful machine. Indeed the creation of a large number of agents that run in parallel can be quite greedy in terms of RAM and CPU usage so it is necessary that the movements of the agent is fluid. It means that the treatments must be fast and optimized.

- Data structure needs also to be optimized because a bad data structure can cause some delay on detections for collision detections or movement calculs.

7. Development constraints

We choose the Agile mode by an incremental model and Java as programming language.

- Reliability and fault tolerances : if the simulation detects a fault, it will have to stop. In case there are no fault, it's the user who must stop the simulation.

- The behavior of the system in abnormal situations : in abnormal situations (in case of extreme problem), the simulation have to stop. If it's not an extreme problem : the simulation continues until the user stops.

- Safety : it is not necessary in this simulation project to have a password or a login, because the simulation can be launch by anyone having the code (and the configuration, software necessary).

8. References

Courses and TDs

https://repast.github.io/

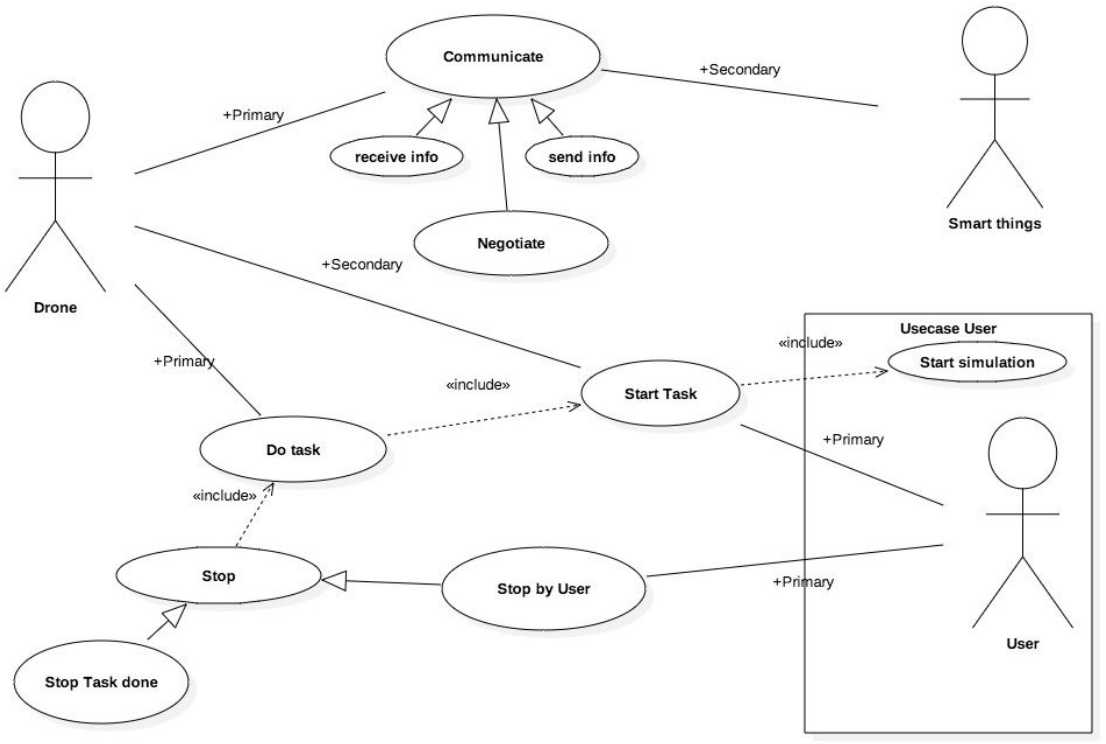
https://www.java.com/fr/

9. Index

10. Annex

- Method used : for this project, we used the Agile mode by an incremental model

Use Cases - Drones



Name	***Description***	***Primary Actor***	***Secondary Actor***	***Pre-condition***	***Post-condition***
Starttask	*-The drone will position itself at a different place* *-He will start the task triggered by the user*	*-User*	*-Drone*	***- Assign task***	*-The drone is doing the task*
Do tasks	*-The drone will perform the tasks that the user assigned to it, such as delivering or racing*	*-Drone*		*-Having a task chosen and assigned*	*-Task has been accomplished OR Not*
Communicate (Send / Receive signal)	*-Communicate with others entities*	*-Drone*		*-other smartThing to communicate*	*-the message is sent or received*
Negotiate	*-Contact the Dock Station to*	*-Drone*			*-The dock station*

	know the availability *-Exchange message in order to know which drone may park first*			*-Definition of the conflict*	*availability is known* *-Continue task*
Stop	*-It stops only if it has completed its task(s) or if the user has to stop the simulation*	*-User*		*-One simulation is launched*	*-The simulation is stopped*
Avoid Collisions	*-A drone can avoid collision with smart or non-smart obstacles/objects*	*-Drone*		*- Drone is doing the task*	*-Collision avoided*

- Class diagram of this project in attachment.