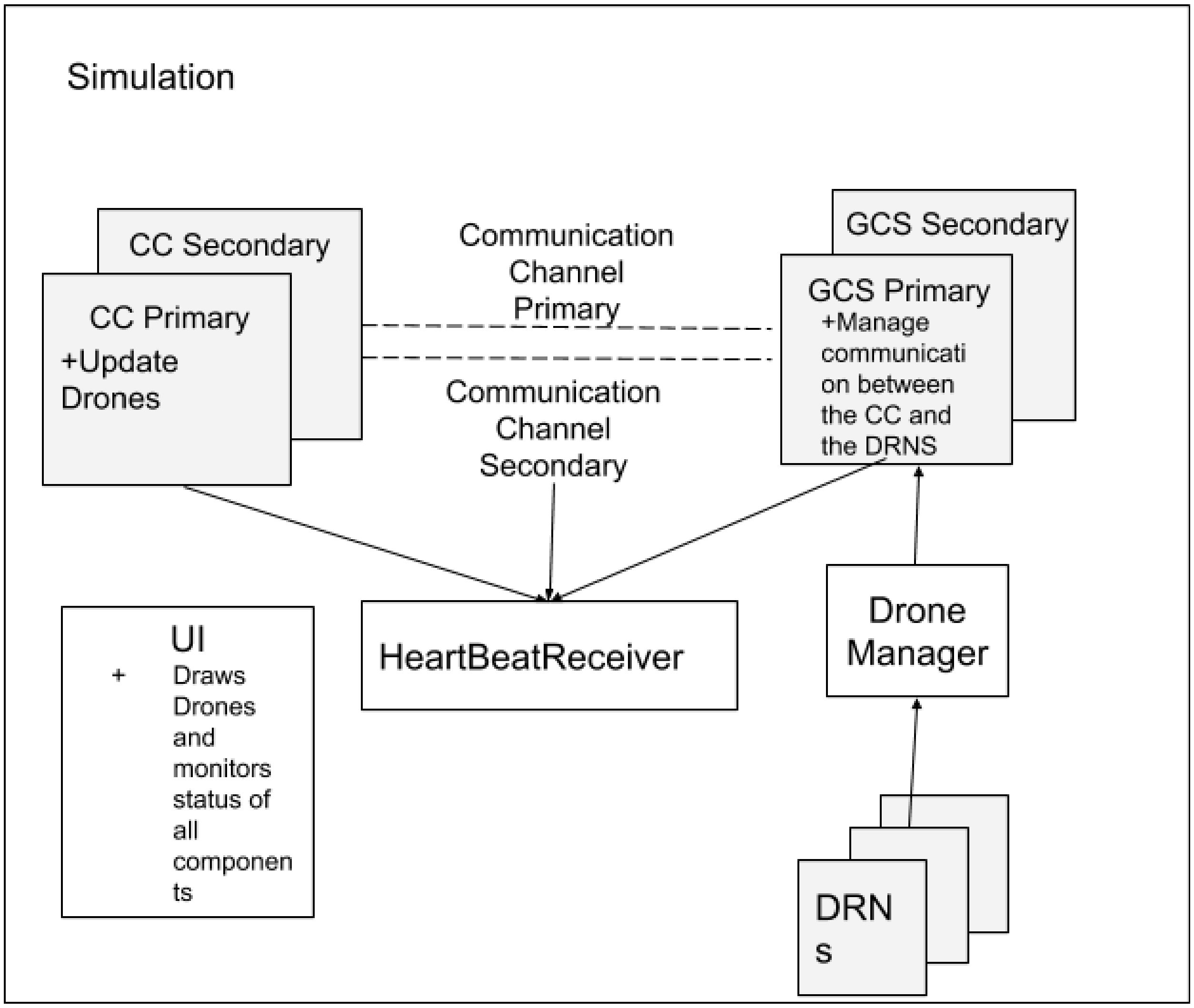
High Availability Design Document

Bryce Badura, Emily Koykka, Taylor Murray

D1.1) Marchitecture


Simulation
CC Secondary
CC Primary
+Update Drones
Communication Channel Primary
Communication Channel Secondary
GCS Secondary
GCS Primary
+Manage communication between the CC and the DRNS
UI
+ Draws Drones and monitors status of all components
HeartBeatReceiver
Drone Manager
DRNs

D1.2) Design Decisions

"A list of architectural design decisions and rationales. This should include an explanation of the tactics you will use, how they will be applied, and explanations for how the combined use of tactics will ultimately achieve high availability"

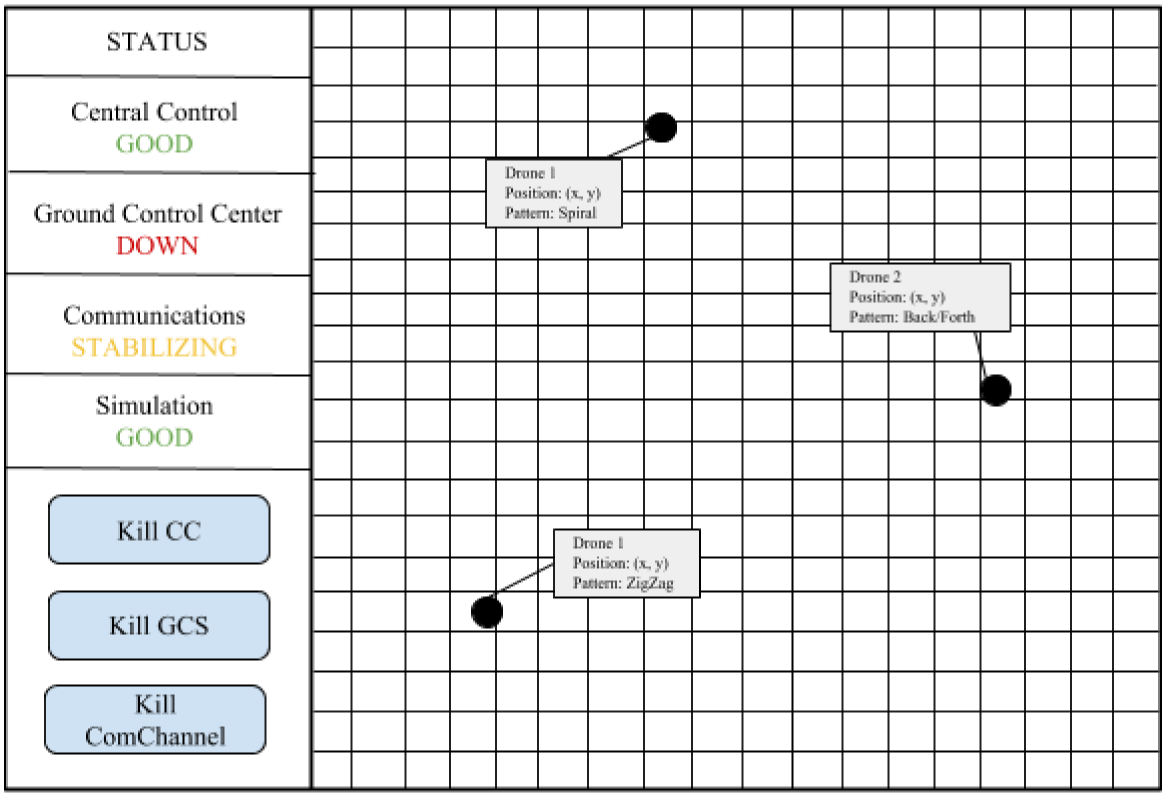
- Fault Detection for the Central Control, Ground Control Station, and Communication Channel

    - To be able to easily monitor the availability of the major components of the system, we will use the Heartbeat fault detection pattern. Each major component will have a HeartbeatSender object that a HeartbeatReceiver module will receive to check on their status and handle exceptions according to error or faults. It also will be used to update the UI elements that report on the status of each of the major components and their redundant components.
- Recovery Preparation and Repair
    - Each system component will use an active redundancy model with hot sparing. This will result in fast recovery time as per the project description. Our rationale for each component is as follows:
        - Central Control: The central control bust batch together the updates for the drones to pass to the communication channel. Utilizing of warm spare poses the risk of the system getting behind on drone updates if we decide to only update the redundant spare node every other update. This will cause a lot of trouble in keeping track of what drone should be headed where because by the time the chaos monkey has deleted the main node, the warm spare will already have obsolete information. The amount of updates required to keep the warm node sufficiently up to date becomes what is equivalent to a hot spare. The same goes for the cold spare, but the problem is worse because the recovery will be much slower.
        - Ground Control Station: The drone manager contained by the GCS will have the same problems as the updates described above with the Central Control. Too much extra logic has to be built in to make sure that the recovery process is actually a full recovery process and that no information has been lost. A hot spare keeping the second node up to speed on all changes will make for the most efficient, straightforward, and fast recovery.
        - Communication Channel: As the Central Control's main source for all information on the drones it needs to update, for the reasons stated above, an active redundancy hot spare approach is necessary.
- Recovery - Reintroduction
    - We are using hot sparing so when the Chaos Monkey brings a component down we will be able to bring its redundant counterpart up immediately. This allows the program to get to work on bring the original component back up. During the transition from failure to reintroduction process the component will be in shadow mode. The system monitor coupled with the heartbeat will detect when a component is back up and when that occurs the system will go back to using it.
- Drone Factory

    - The design of creating drones will follow the factory design pattern such that three drones will be customized through the ground control system and added to the drone manager. The factory will allow specification of the drones' fly patterns and initialize their starting positions.
- Drone Manager
    - The drone manager is simply a data structure that holds of the instantiated drones. There will be an instance of it in the GCS class.
- Drone
    - The drone only needs to hold its flight pattern as a string and its own position. The simplicity here is because the Central Control is in total control of its movement.
- Ground Control Station
    - The ground control station will hold the ingoing and outgoing "message queues" for communication between it and the Central Control. It will also house the Drone Manager so it can easlier get information between the Central Control and the drones.
- Communication channel
    - The communication will be a queue encapsulated as a class. We plan to use a flexible version of the Command pattern to get commands between the Central Control and the drones. This gives the design a way to maintain state for when the communication channel gets brought back up and continues to get messages between the CC and the GCS.
- Chaos Monkey
    - The chaos monkey has access to all system controls and will randomly disable them in order to test for availability and error handling. The chaos monkey will be a thread which sends interrupts to the controller threads, suspending and killing the processes. The chaos monkey will have access to all of the running threads of the program.
    - Each thread will have a method to bring down each component. Since the Communication Channel is not a thread, a flag will be used to cut off the communication channel. Since there is only one Chaos monkey thread we will not have to worry about making this operation thread safe.
- HeartbeatSender
    - The heartbeat sender will have a set interval at which is will send a message to the Heartbeat receiver. Each of the GCS , Central Control and the communication channel will contain an instance of one.
- HeartbeatReceiver
    - The heartbeat receiver needs a method to report on the status of each component so the UI monitor can be updated

- Simulation
    - Since the simulation class will manage all the communication between all components. It is where the recovery methods will be. It will also get the information for drawing drones to the UI. This is so each component of the simulation can focus on just its function.
- UI
    - The UI will draw each of the monitor updates and drones to the drones
    - The handlers for the buttons will call on the Chaos monkeys methods to kill the corresponding component. While the component is down the other buttons will be unavailable since only one component may be down at a time.

D1.4) UI Mock



D1.5) Breakdown of Tasks
GANNT Chart

Activity	Assignee	Start Time	End TIme	10/30	10/31	11/1	11/2	11/3	11/4	11/5	11/6	11/7	11/8	11/9	11/10	11/11	11/12	11/13
UI Development	Bryce	10/30	11/2	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩							

	Badura			🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩							
Simulation	Bryce Badura	10/30	11/13	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Central Control	Taylor Murray	10/30	11/2	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩							
Ground Control Station	Emily Koyyka	10/30	11/2	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩							
Communication Channel	Emily Koyyka/ Taylor Murray	10/30	11/13	🟨	🟨	🟨	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
HeartBeat Send/Receiver	Taylor Murray	10/30	11/2	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩							
Redundancy Imp.(Hot Spare Integration)	Emily Koyyka/ Taylor Murray	10/30	11/13	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Integration Testing	All	11/4	11/13						🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
High Availability Testing	All	11/2	11/13				🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩

D1.6) High Availability EARS Requirements

- The system **SHALL** be able to operate for 60 seconds without a CC. If a/the CC is not restored within 60 seconds then the Drones will return to land (we estimate this will complete within 120 seconds) – allowing for a total of 180 seconds without a CC.
- **IF** a failure to a GCS occurs, then the system **SHALL** recover by switching to the hot spare node.
- **IF** a failure to a communication channel occurs, then the system **SHALL** recover within 30 seconds via the hot spare node.
- **IF** a failure to a Central Control occurs, then the system **SHALL** recover via the hot spare.

D1.7) Design EARS Definitions

- **IF** the Central Control (CC) is online, then the CC **SHALL** update the the drones position based on their fly pattern.
- **IF** the Ground Control System (GCS) is online, the GCS **SHALL** send the CC then drone’s positions.

- **WHILE** the Communication Channel (Co.Ch.) is online, the Co.Ch. **SHALL** transmit information between the CC and the GCS.
- **WHILE** the CC is online, the drones **SHALL** be updated according to the instructions of the CC.
- **WHEN** the HeartBeatReceiver receives the heartbeat from the HeartBeatSender (of any object) the system **SHALL** continue to operate.
- **IF** the HeartBeatReceiver does not receive the heartbeat from the HeartBeatSender (of any object) the HeartbeatReceiver **SHALL** handle the exception.
- **WHILE** the simulation is running, the UI **SHALL** report the the status of each component and the positions of the drones.
- **WHEN** the the KILL <Component> button on the UI is pressed, **IF** there is not already a component down, the UI **SHALL** send an interrupt to the corresponding thread.