Lost Pet Finder chip/app: Architecture Assignment

CS 361: Software Engineering I

Group 16

Haya Ahmed
Ryan Alcorn
Marc Baiza
Shannon Farazi
Matthew Koenig

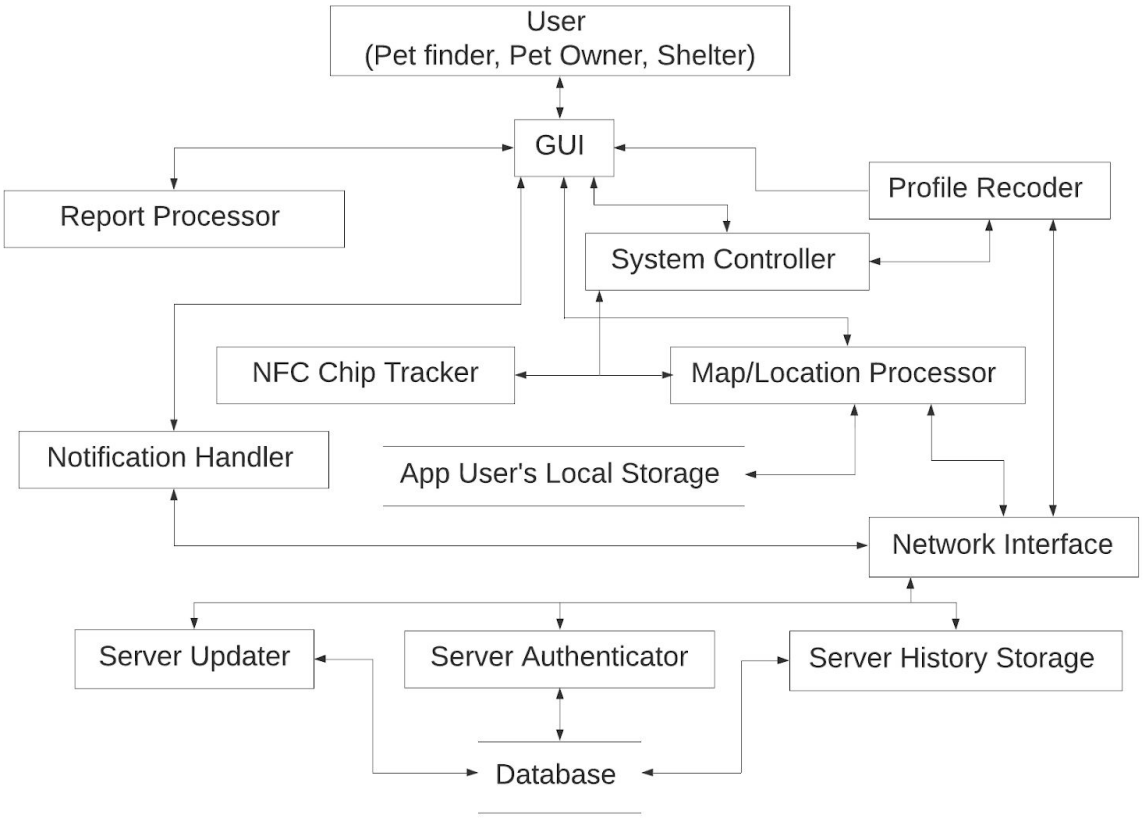
System Architectures

Architecture 1

Data-flow Diagram 1

Typical Layered Architecture

Client Heavy



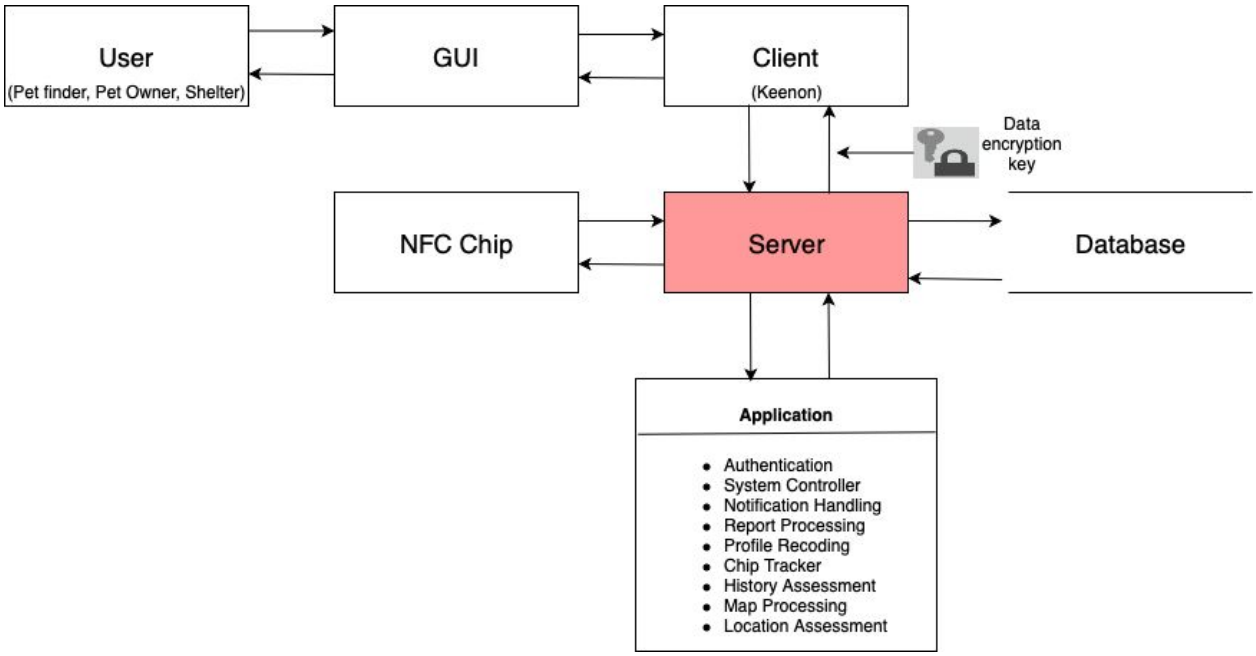
Architecture 2

Data-flow Diagram 2

Typical Repository Architecture

Server Heavy

(Similar to Client-Server Architecture)



Key Quality Attributes

- ## Maintainability
    - Architecture 1
        - User will get a notification that an update is available. If the user continues to use the app some features may be disabled if the user does not update right away. Providing the updates in this way would allow the customer the flexibility to update on their own time, but also prevent unwanted bugs from occurring if the update cannot be performed immediately.

- Architecture 2
    - In a client-server architecture, the server operates as the locus of most changes that would need to occur. This has the advantage of centralizing any updates, but has the disadvantage of requiring maintenance on the server itself, affecting all users ability to utilize the application at all.

Reliability

- Architecture 1
    - This architecture will work under assumed conditions, with this architecture that is less server-heavy we could experience crashes in the system that will temporarily close the application, but would fix rather quickly as the application would re-open, and work as it should.
- Architecture 2
    - It is presumed that, like Architecture 1, this architecture will work under assumed conditions. However, in the event there are failures with the server, the outages will be significant in nature, more so than any issues in Architecture 1.

Integrity

- Architecture 1
    - For security reasons if a chip is scanned, that is not registered to an animal this could provide issues as the architecture currently has no authentication process before moving to the processing and location stage. Even though this issue will not occur often it is a point of discussion.
- Architecture 2
    - Using the same hypothetical as above, this architecture would prevent the system from going into a bad state because the application would first authenticate that the scanned chip belonged to a registered animal before moving on to further steps. Any failed authentication would simply result in an error message.

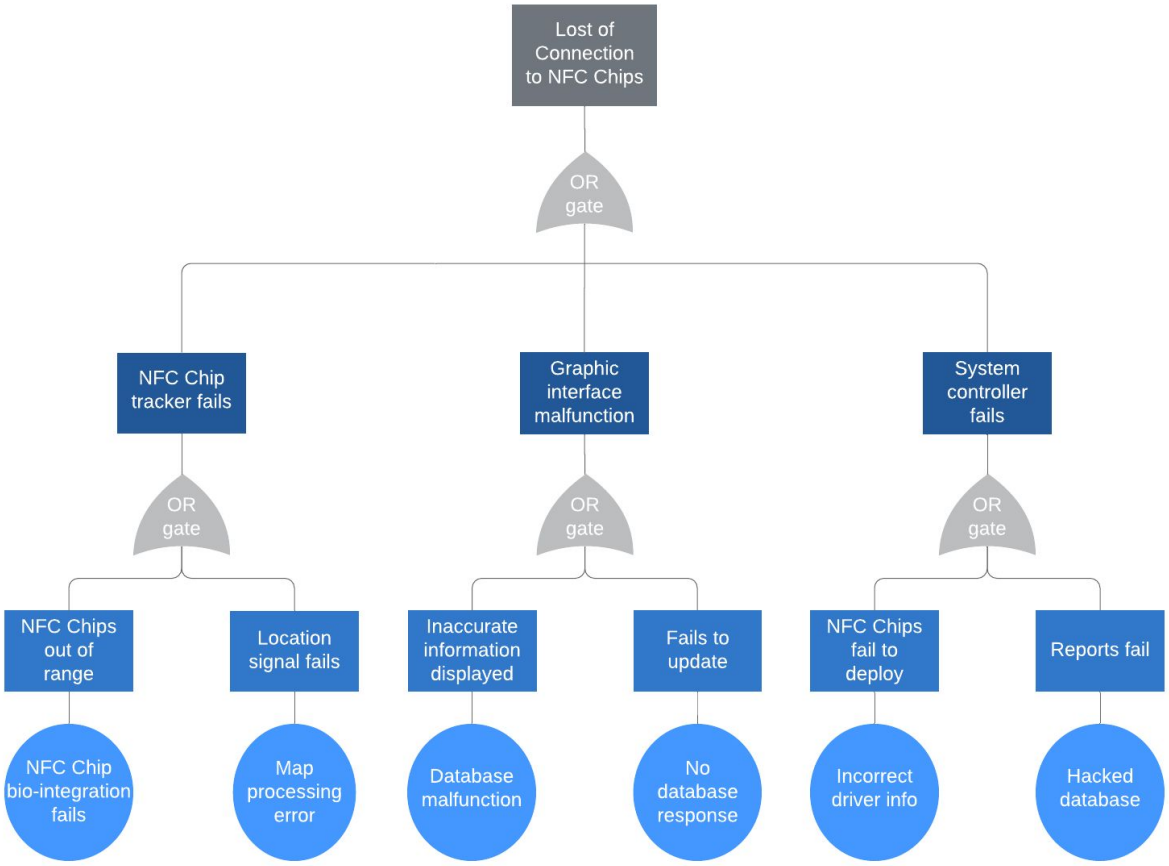
- ## Portability
    - Architecture 1
        - Whether the application is designed for iOS or Android, it will be portable to the other, though this architecture is more client-heavy thus, more redesign work on the platform may be needed as there may be platform specific resources that engineers will have to work around. This could also decrease consistency throughout the different applications which in the end could make it more confusing for customers who use different platforms.
    - Architecture 2
        - Portability from platform to platform will be easier than Architecture 1 owing to the fact the server, the focus of the design, will need minimal changes. The majority of the changes will occur in the client, which is a much smaller part.
- ## Rusability
    - Architecture 1
        - This architecture would provide useful if designing another type of GPS-style tracking system thus, could be reused in another system with similar steps. User use cases may change, and the microchip specifications may be changed, but the rest of the system could easily be ported to a new system with a different idea in mind.

    - Architecture 2
        - This architecture has limited reusability. The server-heavy design is built for the specific event(s) the customer has requested. The database would be virtually unusable, as would the notification system, among other features.

Failure Modes

Loss of Connection to Chips

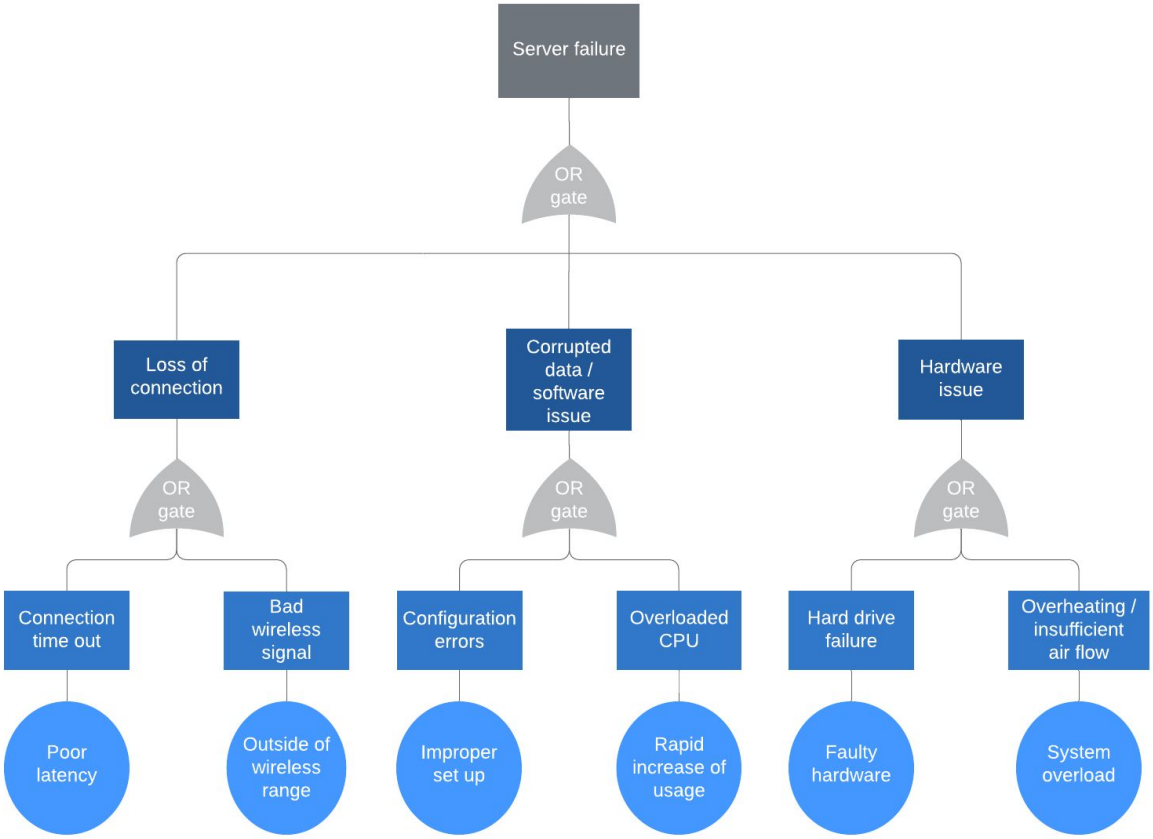
Fault Tree



For loss of connection to NFC chips, architecture one and two are affected equally. Both architectures could still deliver a partially working app since this kind of failure only affects location assessment of pets. However, one can conclude that architecture two is still more prone to this failure than architecture one. This can be explained by emphasizing on the fact that architecture one is more layered than architecture two. More layering ultimately would lead to sufficient amount of error prevention and recovery. There should be precautions to not allow such failure to occur since it would be a setback in the app's mission.

Server Failure

Fault Tree

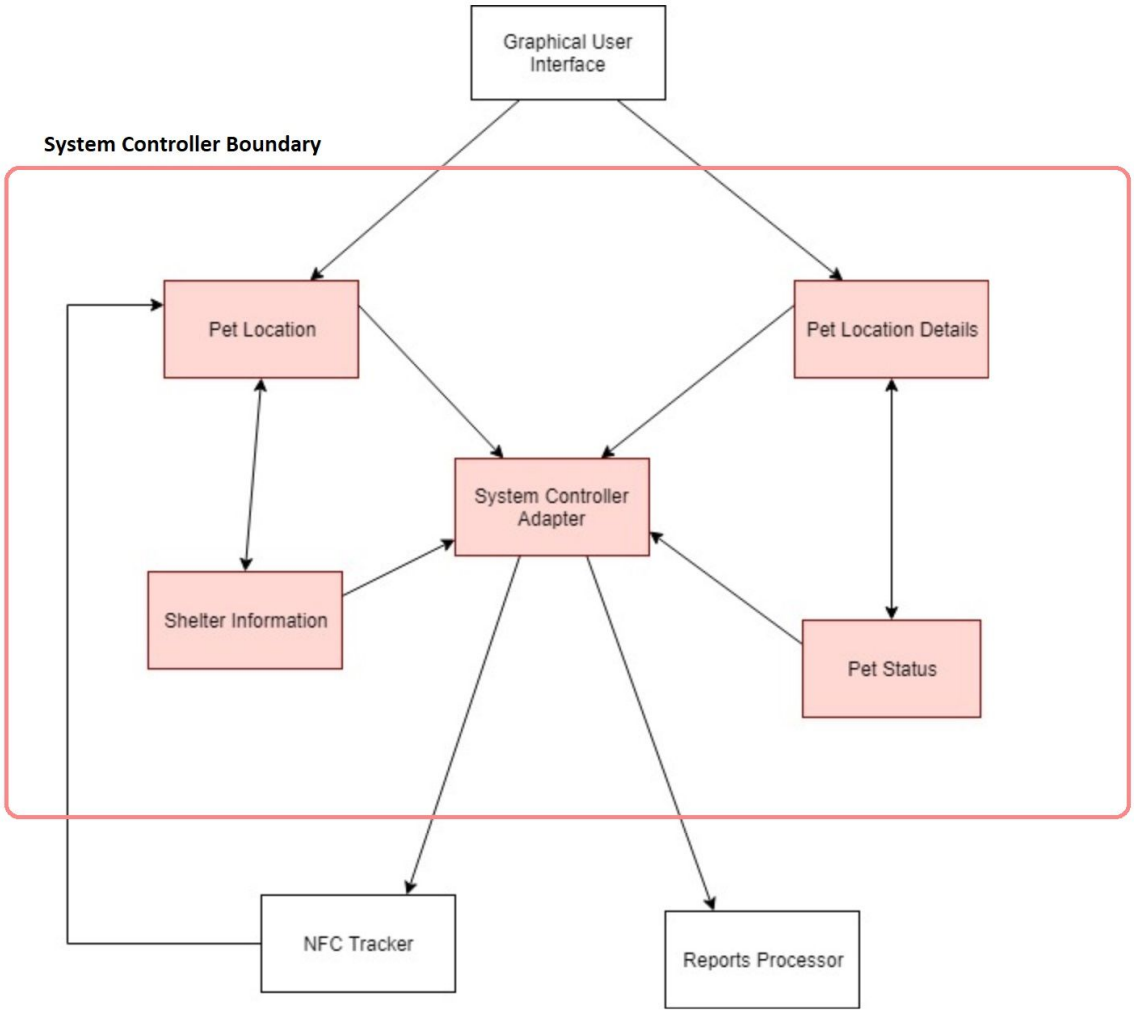


The architecture two is the most failure prone for the server failure. The reason being if this failure occurs, architecture one utilizes a secondary local database where a copy of backup for individual account data would exit to temporarily allow at least a limited use of app possible. Architecture two utilizes a central database access, which would lead to complete loss of app usage if the server failure occurs. The user would be unable to retrieve new data and update the existing data globally. The recovery from this failure, though costly, would be achievable by repairing the servers. This way the promise of the app to assist in bringing the lost pet home is still viable.

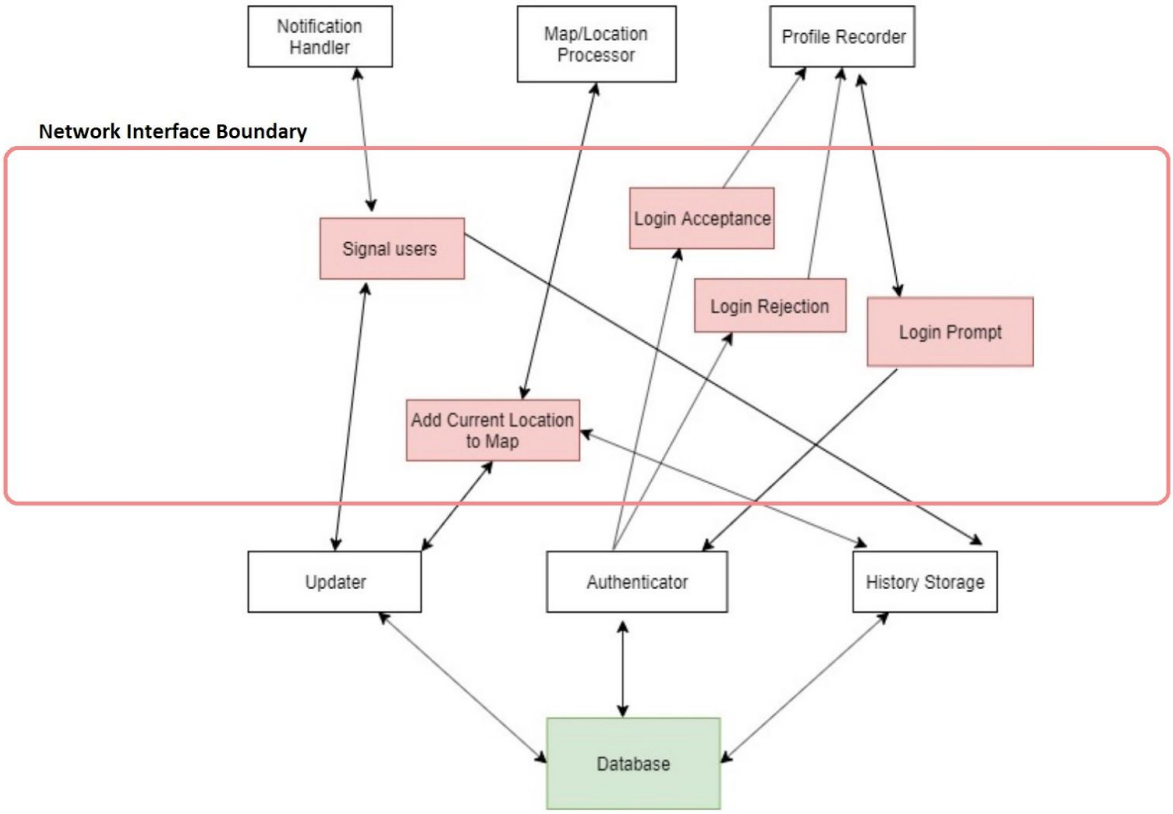
Architecture Decomposition

Important Element 1

System Controller



Important Element 2



Use Case Walkthroughs

Use Case 1: Reporting a missing pet to those in the vicinity

- Architecture 1 is a layered architecture meaning that components provide services to the next layer and they hide the lower layers. Use Case 1 deals with reporting a pet missing to those in the vicinity through the app.
- Based on the layout of our architecture, a user(owner) going through the flow of events in Use Case 1 will only see the GUI which presents the user(owner) with different pages and options in the app. The GUI will be connected to other important components in the system.
- The user(owner) will first attempt to log in to the app through a server which contains a database that stores the user's(owner's) information. A Server Authenticator will help verify the user's(owner's) credentials. The database will be updated through the Server Updater and the Server History Storage will contain information about current/past pet alerts. The Network Interface will allow communication between the server and other components at higher levels.
- The Server/Network Interface will send profile information back to the user(owner) through the Profile Recorder upon verification of their account which will be presented to the user through the GUI.
- The user(owner) will detail all known information regarding the last known location of the pet and send it through the GUI to the Report Processor. The Report Processor will convert the data into information other components can use. The Report Processor will send the converted report back to the GUI to display to the user and then the GUI will send the information to the System Controller, Notification Handler, and Map/Location Processor.
- The System Controller will send that information through the Profile Recorder, which changes the data into information the server can understand, to the Network Interface/server. The server will store the report in the Server History Storage.
- The Map/Location Processor will receive tracking information from the NFC Chip Tracker and will populate the map with a pin location of the missing pet. The information used in Map/Location Processor will also be stored on the App User's Local Storage in case of system failure.
- Once the server processes the missing pet alert, the Notification Handler will send out a push notification to other users GUI's.
- Other users will receive information like the pet's current location on the map through the Notification Handler which comes from the Map/Location Processor and NFC Chip Tracker by way of the Network Interface.

Use Case 2: Locating / Helping with capture of lost pets

- Architecture 1 is a layered architecture meaning that components provide services to the next layer and they hide the lower layers. Use Case 2 involves the location of pets, and the process of helping others retrieve their pet.
- Based on the layout of our architecture, a user(owner) going through the flow of events in Use Case 2 will only see the GUI which presents the user(owner) with different pages and options in the app. The GUI will be connected to other important components in the system.
- The user(owner) will log in to the app through a server which contains a database that stores the user's(owner's) information. A Server Authenticator will help verify the user's(owner's) credentials. The database will be updated through the Server Updater and the Server History Storage will contain information about current/past pet alerts. The Network Interface will allow communication between the server and other components at higher levels.
- The user(owner) will then locate their pet at the location provided on the map on the GUI. This location is provided by the NFC Chip Tracker which then sends the tracking information to the Map/Location Processor. The Map/Location Processor then pins the location on the GUI.
- The user(owner) can then either retrieve his pet himself or send out a push notification to others near the pet to ask for help. This process is done through the GUI, and then sent to the Notification Handler which will send a push notification to all nearby using the Network Interface. This will also interface with the Profile Recorder as before sending out a push notification the pet must be in a “lost” status.
- Once the pet is retrieved by the user(owner) the GUI will interface with the Profile Recoder, System Controller and the NFC Chip Tracker to scan the pet, and reset the profile from “lost” to normal status.
- If pet is retrieved by a user(not owner), the user will use the GUI and it will interface with the Notification Handler, System Controller, NFC Chip Tracker, and Profile Recorder in order to change the pets status from “lost” to “found” then instantly send the user(owner) the location of the pet and the information of the person who retrieved the pet.

Use Case 3: Notifying/Reporting pet to local shelters

- Architecture 1 is a layered architecture meaning that components provide services to the next layer and they hide the lower layers. Use Case 3 deals with notifying/reporting missing pets to local shelters.
- Based on the layout of our Architecture, a user(owner) going through the flow of events in Use Case 3 will only see the GUI which presents the user(owner) with different pages and options in the app. The GUI will be connected to other important components in the system.
- The user(owner) will perform all of the steps outlined in Use Case 1 including sending a report of their missing pet to the Notification Handler along with finding tracking data through the NFC Chip Tracker.
- Users(other app users and shelters) will be notified of the missing pet from their Notification Handler and the information/location of the missing pet will be displayed on their GUI. The Notification Handler will receive information from both the Map/Location Processor and NFC Chip Tracker through the Network Interface. The other app users will also receive information about the pet from the Database by way of the Network Interface.
- Once found, the missing pet will be taken to a shelter. The shelter will scan the pet through their GUI and send the information to the System Controller. The Profile Re-coder will convert the data from the scan into information the server can understand. The pet's identity will be checked by the Database in the Server through the Network Interface.
- Once the pet's identity is verified, the user(shelter) will notify the user(owner) by way of the Notification Handler to be displayed on the user(owner's) GUI.
- The user(owner) will know the location of the found pet with help from the Map/Location Processor which is connected to their GUI.
- The user(owner) will then retrieve their pet.
- The user(owner) will change the status of the pet to “Found” in the Server by way of the Server Updater. The Database will update the status of the pet and the Server History Storage will record the event for a certain amount of time.

Implication

Our current iteration of client-heavy architecture (Architecture 1) has the GUI sending information to the Report Processor, Notification Handler, and System Controller. An alternative architecture model is that instead of GUI sending data directly to the components such as the Notification Handler, and Map/Location Processor, it would send its data to the System Controller. The System Controller would then direct the packaged data to the respective destinations. The System Controller would also receive data in the form of a report from the Report Processor, which it was previously not connected to. The benefits of this modification stems from the fact that it would simplify design and we could build a robust and secure distributing system around the System Controller.

This alteration would improve reliability and be conducive to better system integrity, as it would allow us to build a robust security system around the information going through system controller. In our current system, in order to add these security measures we would have to add them at each point after the GUI. Thus, we would have multiple points of security to manage and maintain, which may not be as reliable individually when compared to fewer points of robust security.

This alteration would also improve the performance of the system because it will now have a central storage area of data going to and from the GUI. The System Controller will work with components at lower levels instead of having the GUI work with components at those levels in the system. This would cause those components actions to be hidden from the GUI, which is a form of encapsulation, and speed up the processes performed by the GUI. This alteration will also speed up the process of the app in terms of memory being used by the system because the GUI won't have to send data to other components besides the System Controller. The System Controller will handle where the data is being sent and the GUI can deal with app features like displaying information about the pet or showing the map of missing pets in the area.

The potential for failure would be if the System Controller fails. This would leave the GUI being not connected to other components at lower levels like the Report Processor and Notification Handler. This is a common point of failure when one component is performing many other processes, but if the designers make the System Controller more robust then the chance of failure will be lessened.The Database would also still be storing the information received by the System Controller through the Network Interface. Overall, we feel this would be an efficient way to revise the current selected architecture even if it has its own potential for failure.

Team Member Contributions:

- Haya: Decomposition, Implications
- Matt: Quality Attributes, Decomposition
- Marc: Quality Attributes, Use Cases
- Ryan: Use Cases, Implications
- Shannon: Data Flow Diagrams, Failure Modes
- All: Strategy and Planning, Editing