Package Delivery System
Phase #2

Members: Elizabeth Dayton, Caroline Smith, Paul Poe, Collier Robinson, and Adam Bostwick

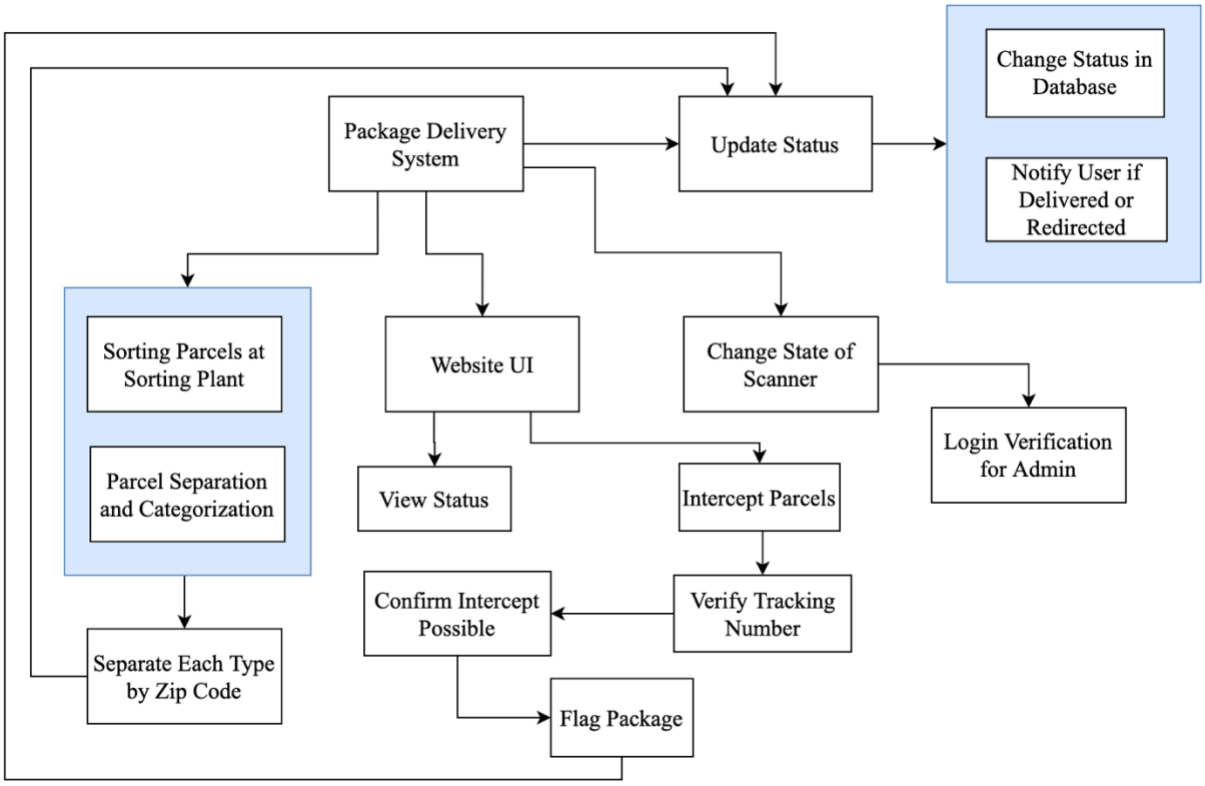
Table of Contents

Phase #1 Changes

The updates we made to our phase 1 documentation include some domain class changes and some design class changes. We got rid of the Flats class, because since flats do not have tracking numbers and cannot be tracked or intercepted, it was unnecessary. We also put all of the functionality related to the user manipulating the package (intercepting or tracking) into a single class called WebsiteBoundary instead of having them randomly separated into various domain classes.

The final update we made was combining the two controller classes into a single controller class, because we thought having separate controller classes was redundant.

Architectural Design



We chose the “Pipe and Filter” architecture for our Architectural Design. We chose this because our system is most closely represented by it. Our system has data flowing from one system to the next and processes data in separate stages. The pros for selecting this architecture style is that it’s easy to understand and evolve. It also supports easy transformations and reuse. The workflow style matches closely how the system would work in real life for a business. It can also be implemented in a sequential or concurrent system. The cons would be that the format for data transfer must be agreed upon between each communicating transformation. Also, each transformation must parse the input and outputs, increasing system overhead. It may not be possible to reuse functional transformations that use incompatible data structures. It might also not be suitable for interactive systems.

Detailed Design

A. Interaction Design

I. System Operations

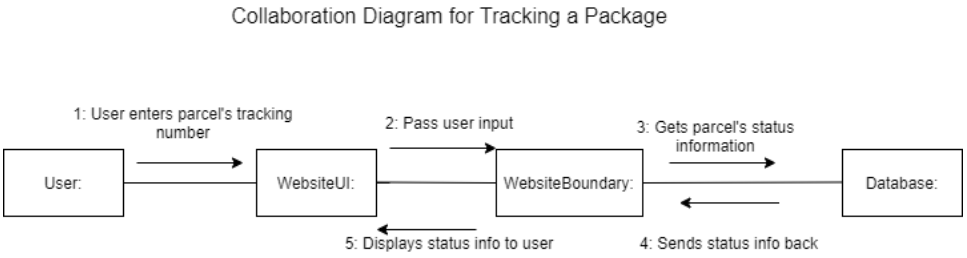
- **Tracking a Package:**
  - checkTrackingNumber(string trackingNum): When a user enters a tracking number into the website UI, the system will need to verify that a package with a tracking number is in the database. If it is not found in the database, this operation will return some sort of indicator like a negative integer value or a false Boolean so that the UI can display an error message.
  - getStatuses(string trackingNum): If a package with the specified tracking number is found in the database, the system will get all the past and current statuses of the package in order from most recent to less recent and display them in a stack-like display (most recent status on top, least recent on the bottom).
- **Intercepting a Package:**
  - verifyInterception(string trackingNum, int interceptType): If the user requests that a package be intercepted, the system must first verify that a package is viable for interception. It must check that the tracking number is in the database and that it is not already out for delivery.
  - flagIntercept(string trackingNum, int interceptType): Once a package has been approved for interception, it will need to be flagged as such in the database. If it has been approved for being held for pickup at the post office, the "Hold" flag will be updated. Alternatively, if it has been approved for redirection, the "Redirect" flag will be updated as well as the new destination address that was entered by the user in the UI.

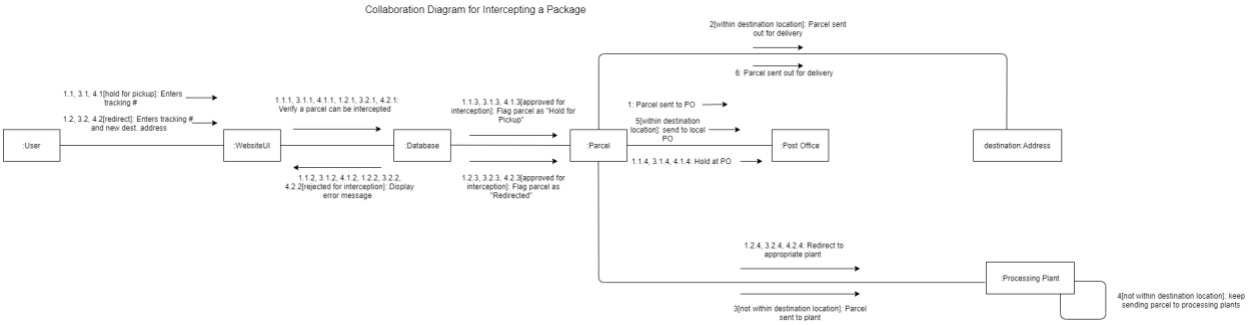
II. Operation Contracts

- **Use Case: Tracking a Package:**
  - **Operation: checkTrackingNumber(string trackingNum)**
    - Pre-condition(s):
      - A package with a specified tracking number must be in the database.
    - Post-condition(s): Nothing changes within the system after a tracking number is checked; however, this operation should return a Boolean value indicating whether the tracking number was found in the database or not.
  - **Operation: getStatuses(string trackingNum)**
    - Pre-condition(s):
      - A package with a specified tracking number must be in the database.
    - Post-condition(s): Nothing with the system changes after this operation is performed, but the website UI will display the past and current statuses of the package.

- **Use Case: Intercepting a Package:**
  - **Operation: verifyInterception(string trackingNum, int interceptType)**
    - Pre-condition(s):
      - A package with a specified tracking number must be in the database.
      - The current state of a package cannot be "Out for Delivery."
      - The dimensions of a package cannot be longer than 108 inches.
    - Post-condition(s):
      - If approved for interception, this operation should return true or integer 1 to indicate it has been verified.
      - If rejected, this operation should return false or integer -1 to indicate it has not been verified.
  - **Operation: flagIntercept(string trackingNum, int interceptType)**
    - Pre-condition(s):
      - A package must be approved for interception
    - Post-condition(s):
      - This operation should flag the specified package as intercepted in the database (either "Hold for pickup" or "Redirect").

III. Collaboration Diagrams

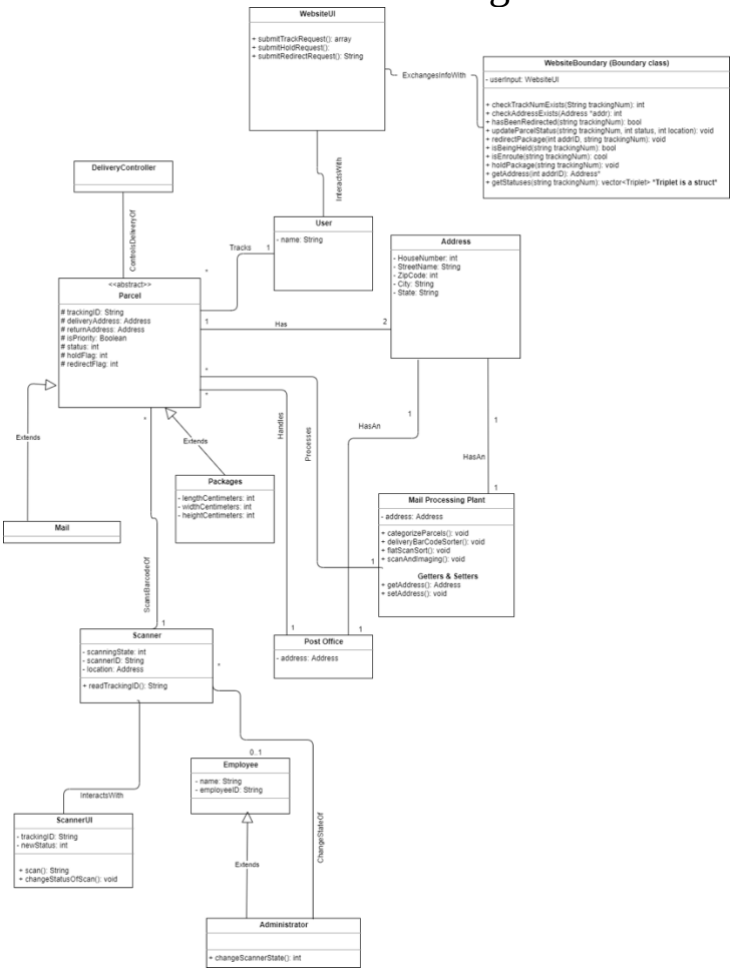




IV. GRASP Design Patterns

- **Tracking a package**
  - This collaboration diagram includes a creator class – the WebsiteUI. It is responsible for creating class instances including Address and WebsiteBoundary, and passing the instances or instance data to the WebsiteBoundary class. This diagram also contains examples of High Cohesion, because WebsiteUI handles user input to the database via WebsiteBoundary. This is similar to how we would have a ScannerBoundary class to handle database calls related to ScannerUI if we were implementing that in our project. In this diagram, Parcel is the information expert, because it is responsible for knowing its current status, its most recent location, priority status, destination return address, and its status of interception. There is also polymorphism in this diagram, because Parcel is a parent class of 2 types: Mail and Package.
- **Intercepting a package**
  - This diagram also contains examples of High Cohesion for the same reasons explained in the previous paragraph. This diagram also contains the same Information Expert as in the previous diagram, Parcel. This diagram contains a few more Information experts, including Post Office and Processing Plant; both of these classes know their own address, which is important when displaying the statuses of a package to the user so they can see where the package is located. This diagram additionally contains Polymorphism because a Parcel is a parent class of Mail and Package.

B. Class Design



C. Behavioral Design

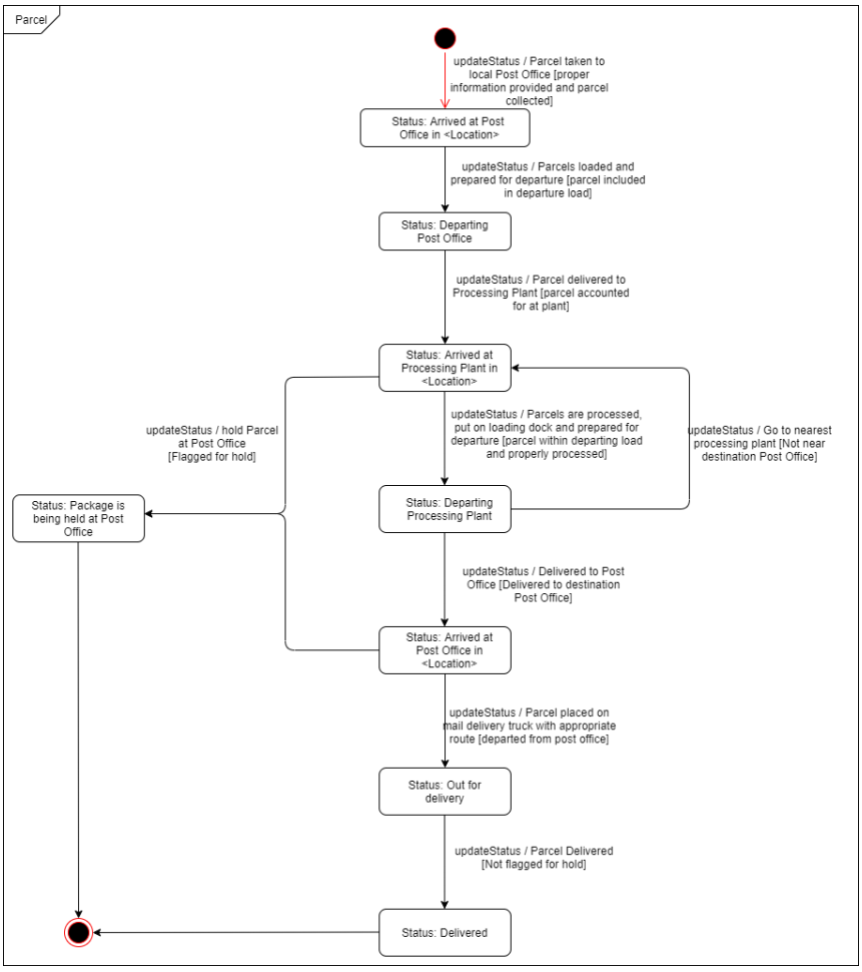
I. Pre/Post Conditions

Classes with State Dependent Behavior:

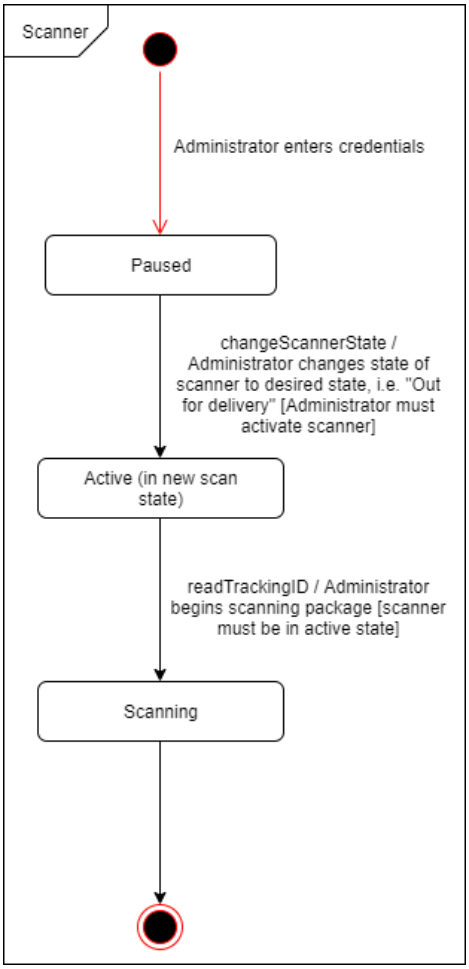
- Parcel
  - updateStatus
    - Precondition – in order for a change of state, the parcel must successfully transition into the next stage of its transit.
    - Postcondition – The parcel's status in the database will be updated to the appropriate status.
- Scanner
  - readTrackingID
    - Precondition – The scanner must be active and have been initialized by an Administrator, as well as the package a proper ID to be scanned.
    - Postcondition – Once the parcel ID has been scanned, the scanner updates the status of the package in the system.

II. UML State Chart

Parcel State Chart:



Scanner State Chart:



III. Procedural Behavioral Specification

Pseudo Code for Parcel:

```
updateStatus(trackingNumber, status) {

        parcel = getParcel(trackingNumber);
        parcel.setStatus(status);

}
```

Pseudo Code for Scanner:

```
readTrackingID() {

        getTrackingID();
}
```