Phase 2: Software Design and Architecture

Group 13: Devante Wilson - 100554361

Shahrukh Zarir - 100489271

Pranav Yadav - 100557540

Tsering Paljor - 100521258

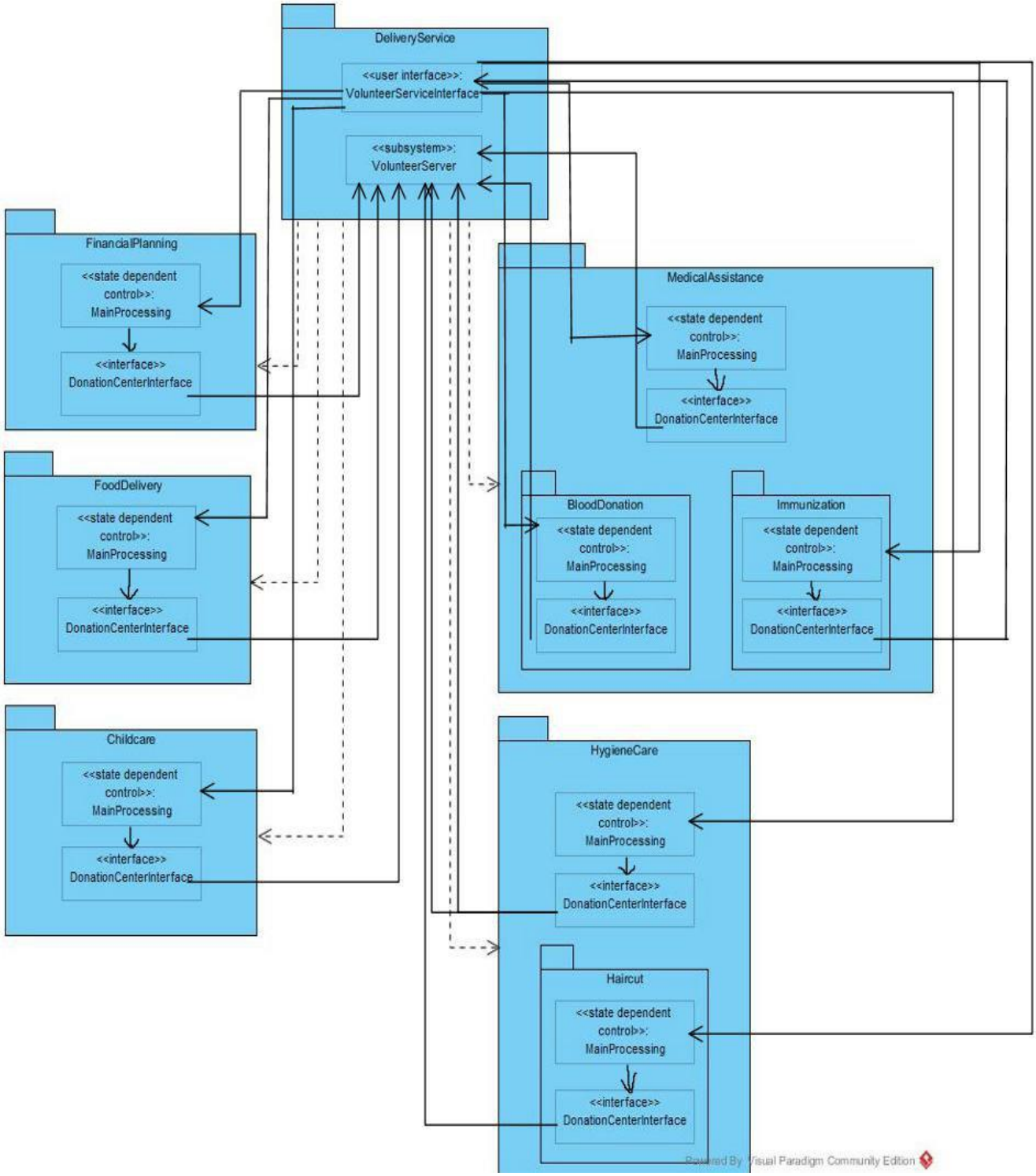
Date: 11/03/2016

Overview & Purpose

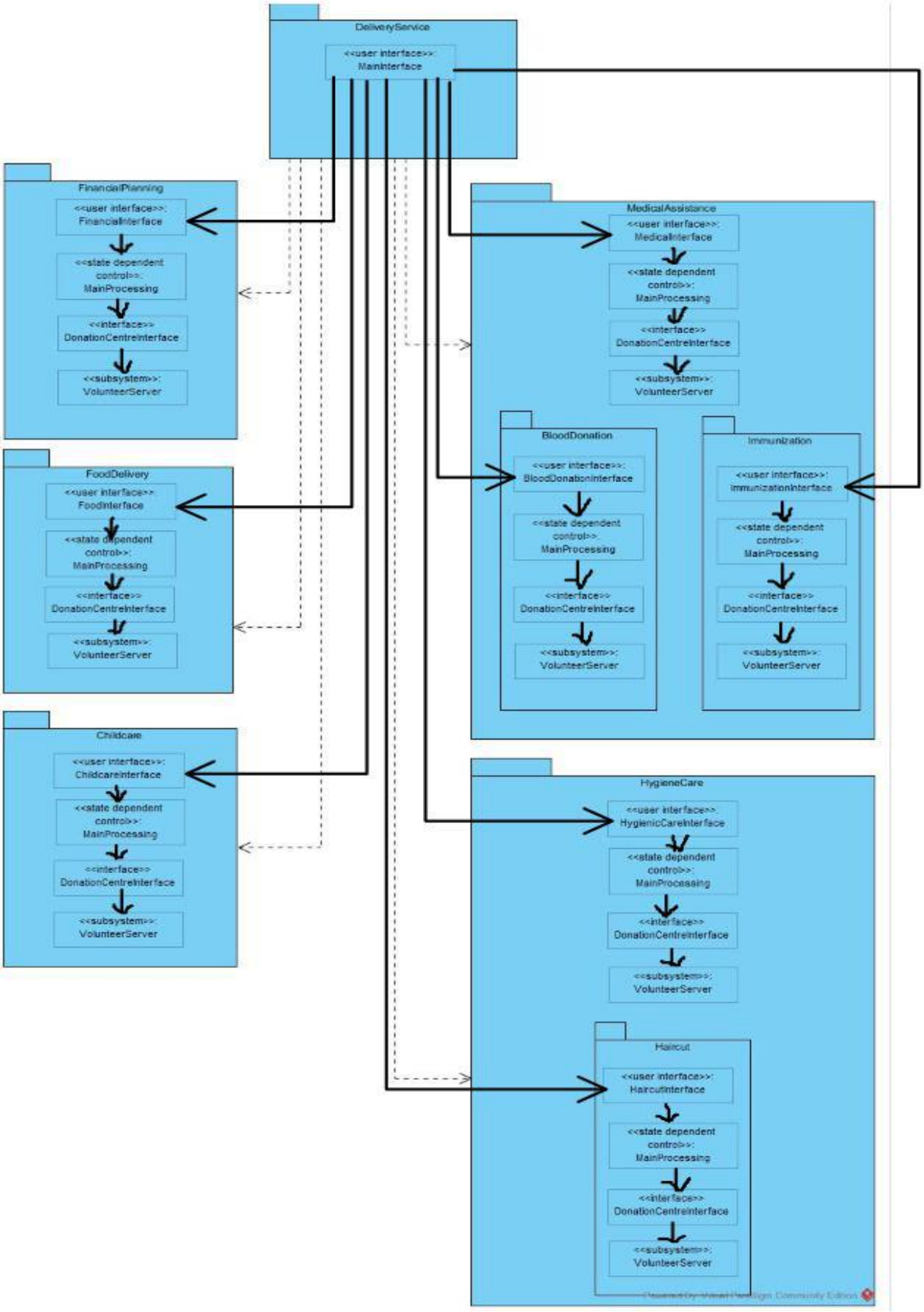
The second phase of the iVolunteer Project will consist of furthering the design process; by incorporating various modelling principles such as dynamic modelling and principles of package cycling, many views of the process can be mapped.

Part 1: Alternative Package Structures

Package Structure 1



Alternative Package Structure (2)



Package Structure Design Rationale

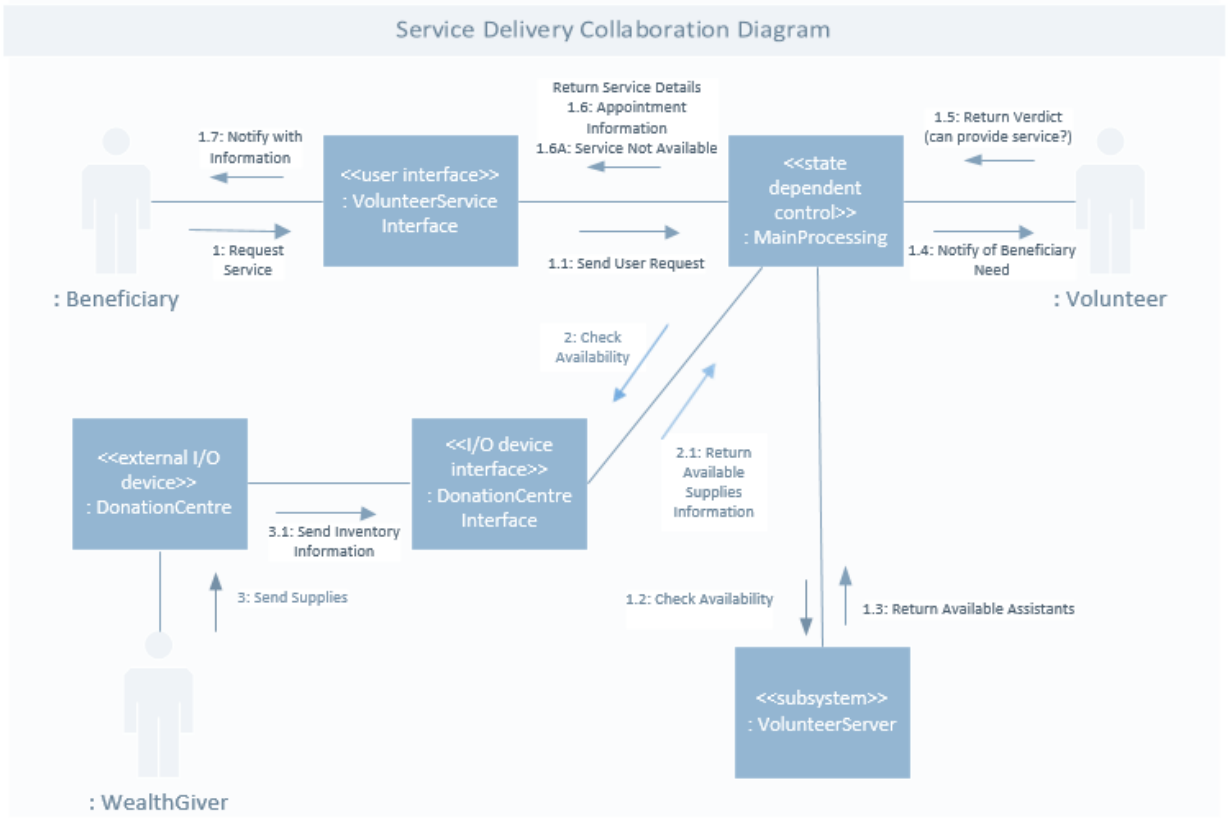
For the design of the iVolunteer project, the team discussed two different ways of designing a package structure to fulfill the requirements in terms of use cases the team has laid out in phase one. The package structures both provide a detailed visual explanation of how the team will approach in designing the project as a whole in a package structure display.

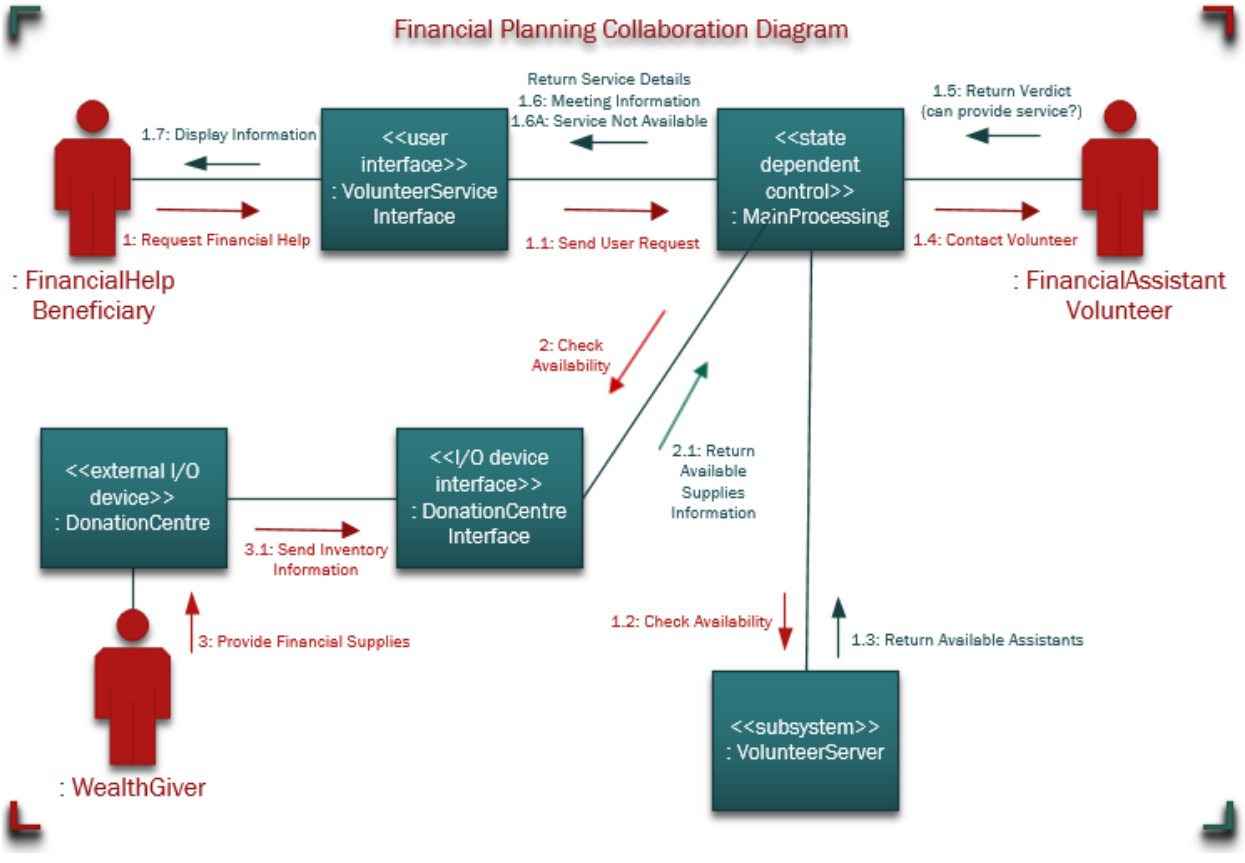
In *Package Structure One* we have designed the packages separating them in terms of use cases. So every use case has its very own package to take care of the processing required in the certain function. Each package representing a use case has a MainProcessing class which is a state dependent control. The MainProcessing class will contain each special detail required for each use case. The reason why the MainProcessing is present for each use case package is because we have a different process for the different use cases provided in the iVolunteer application. We have a similar user interface for all the different use cases so we have provided a general user interface class in the DeliveryService package. This package is very important and contains all the classes that help put all the use cases together. Mainly the user interface and the server to contain all the volunteering information. We did this to create a package, so that when changes are needed to be done the packages below will not be affected. This helps us create a very stable package structure.

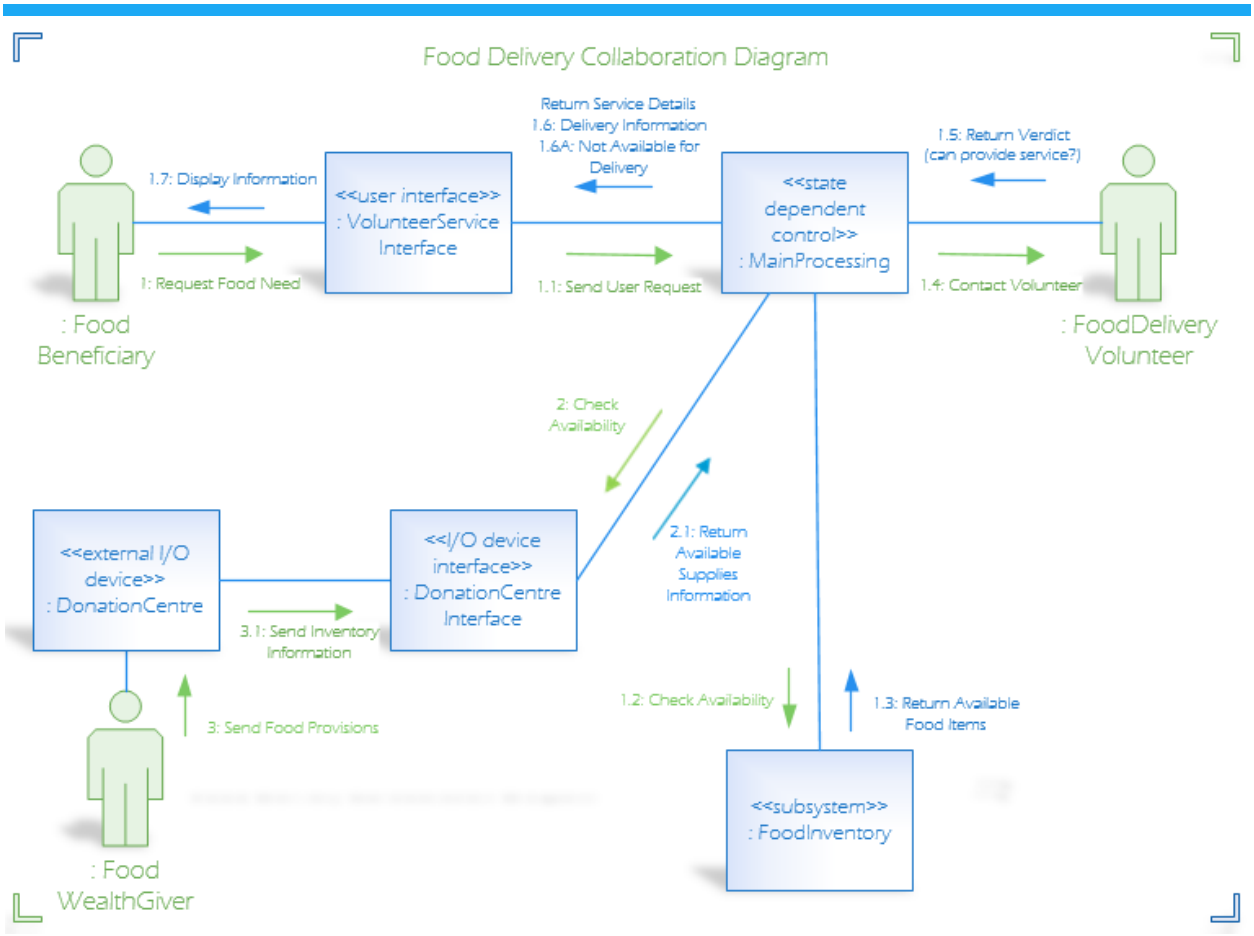
In *Package Structure Two* we decided to make a simple structure but later on figuring out that this design would be unstable and impractical for a final design. This is because there is a presence of quite a few classes that are being repeated within the project. This can be a problem in the future because when changes are required to be made for resolving issue this will create more work for the developers in later patches and bug fixes. For example, our team noticed that in every package representing a use case there was a similarity in the classes present. Each use case package had a different server and a different user interface present. So as final group decision we decided that we would only require one server to contain the volunteering information and one general user interface to handle all the use case functionality.

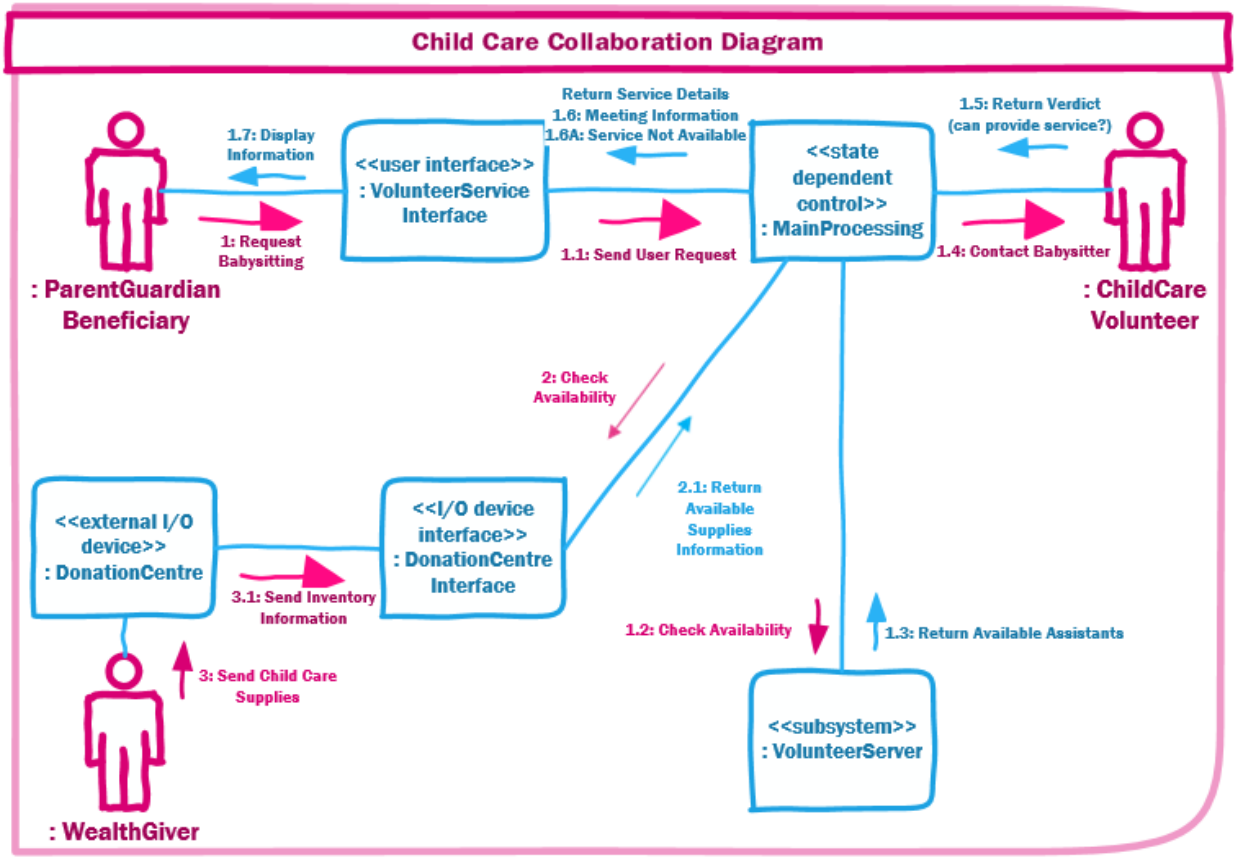
So finally our rationale to choose the first design was mainly because our team wanted to reduce repeated code (although there is more coupling than the alternate package structure) and to avoid any major instable designs that could be affected when making bug fixes and patches.

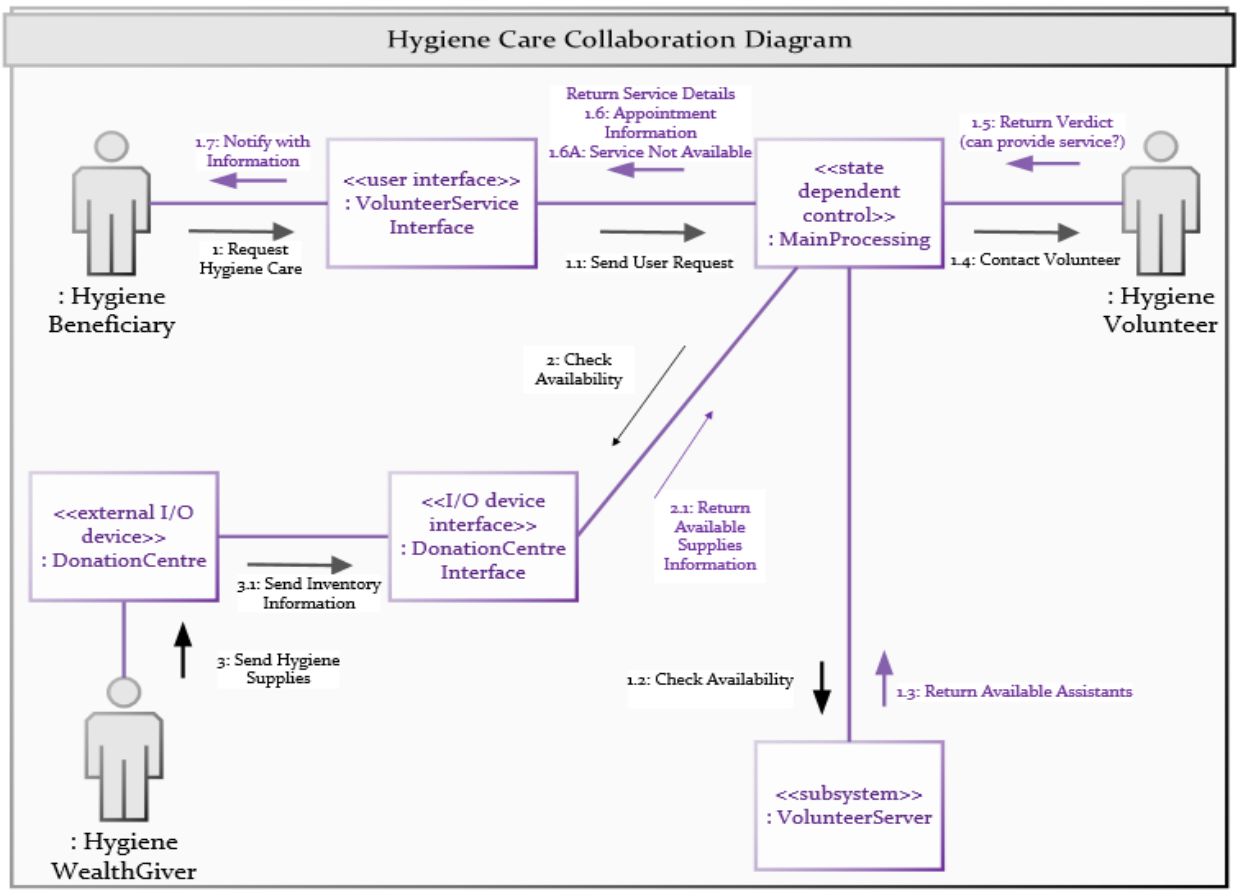
Part 2: Interaction Diagrams

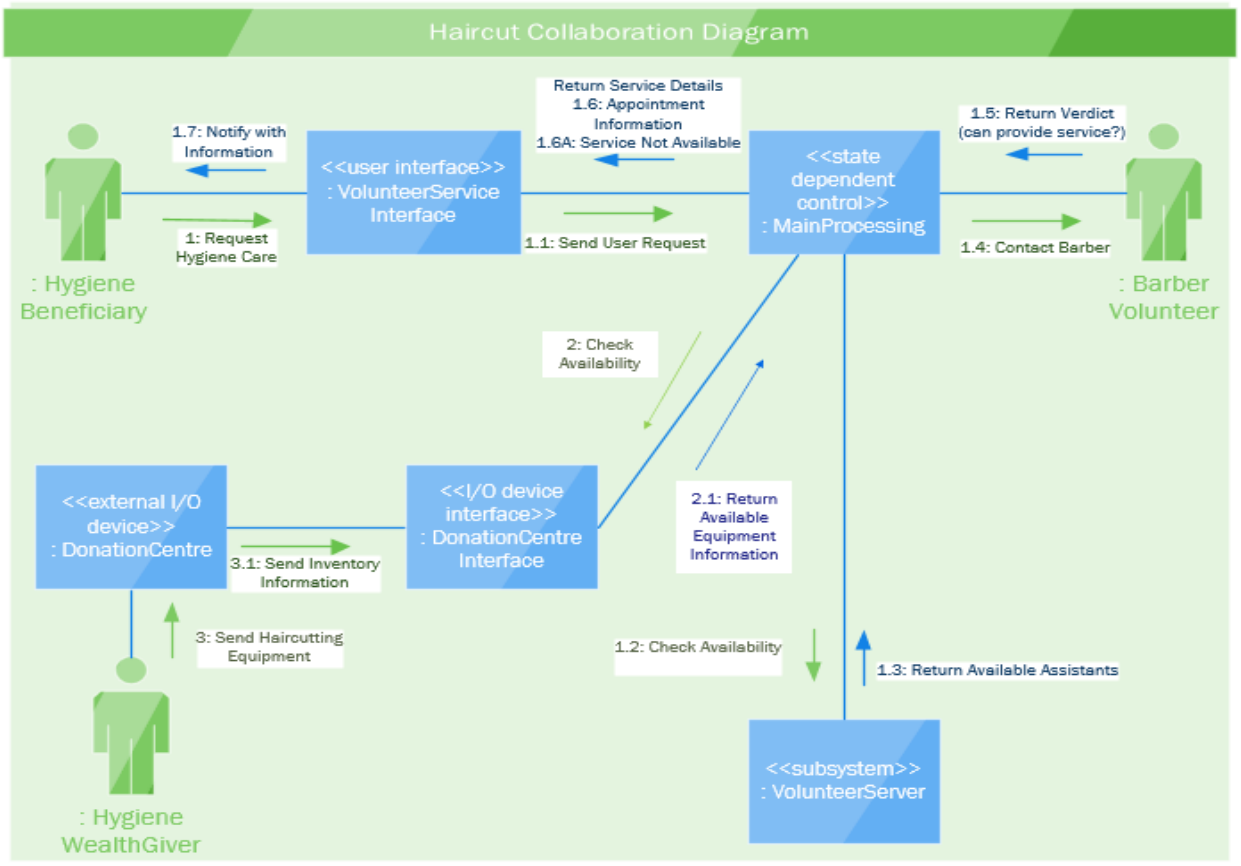


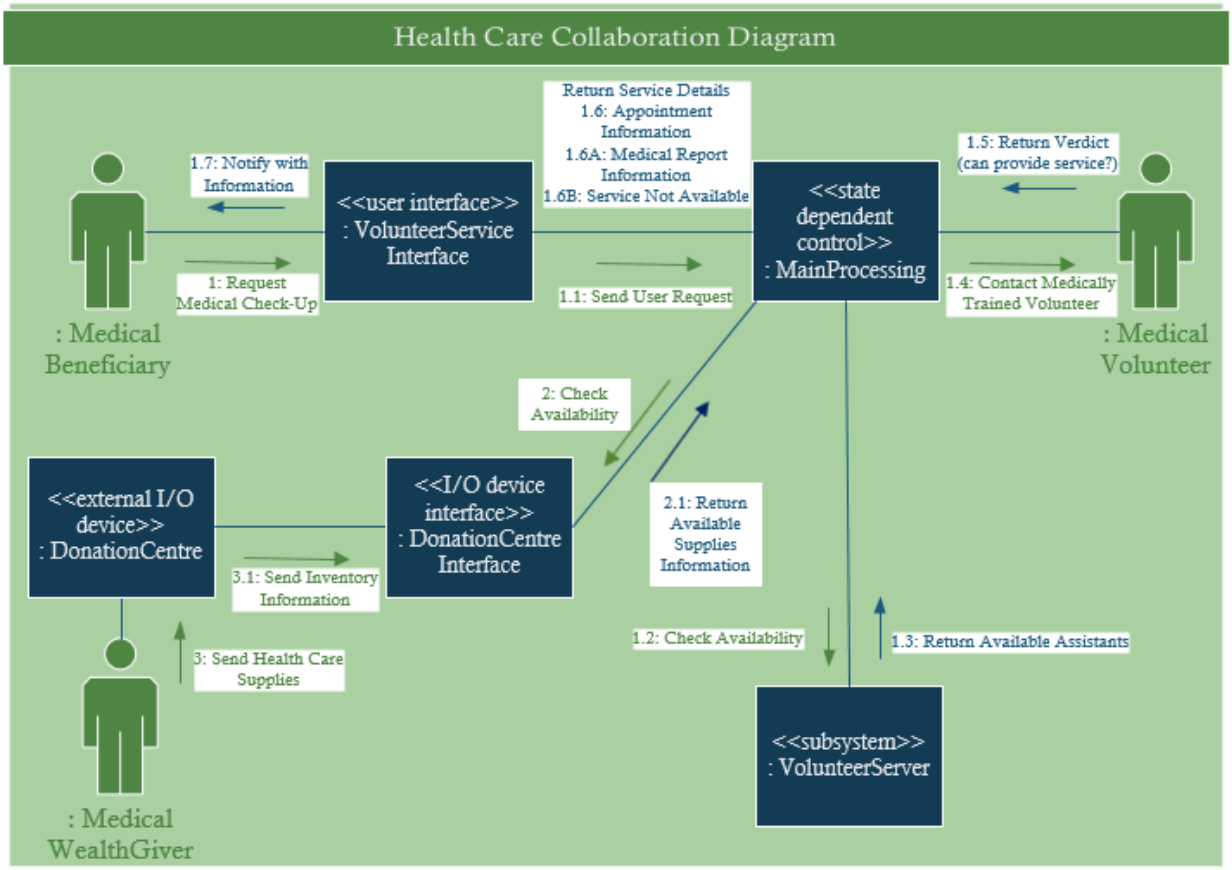


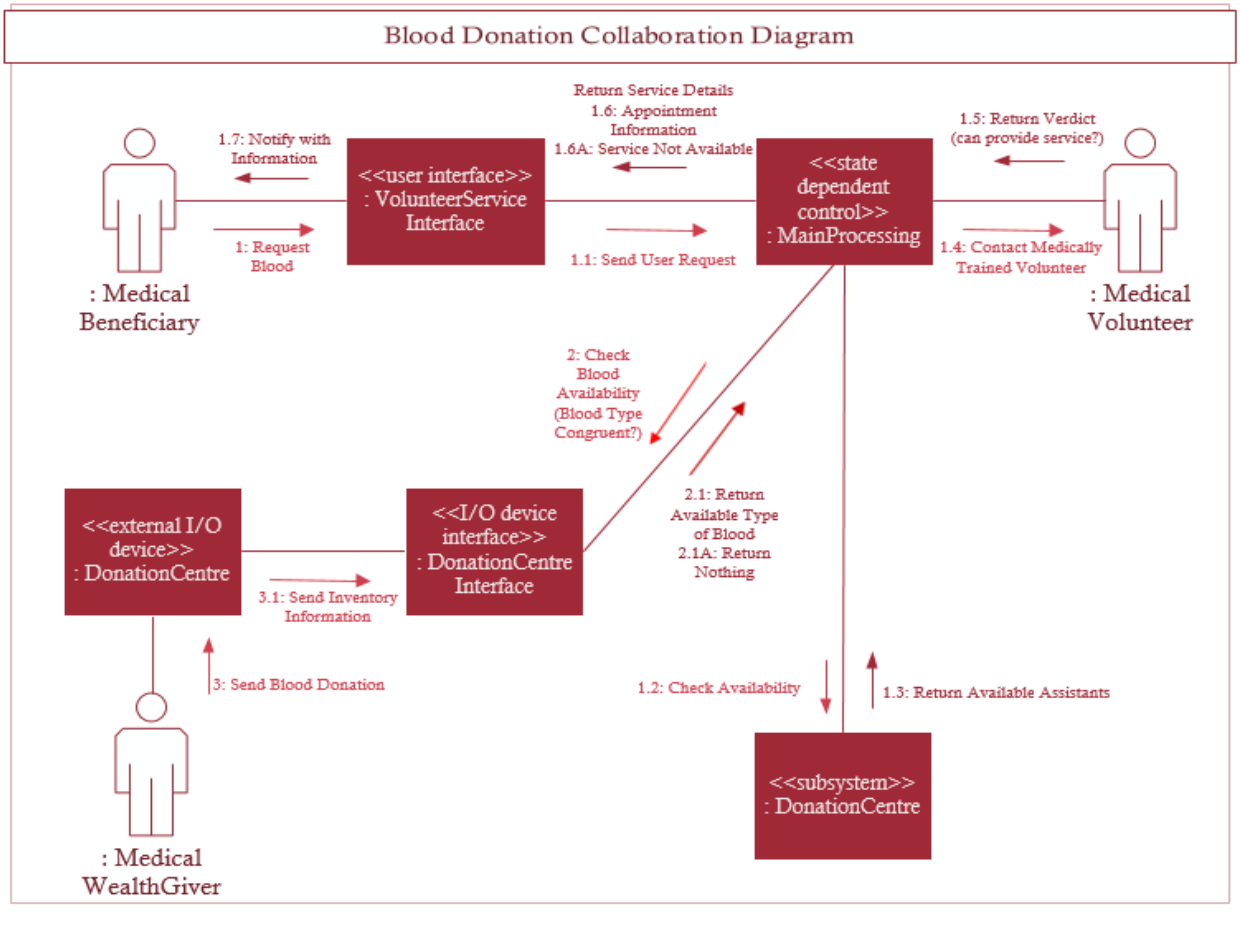

Food Delivery Collaboration Diagram
Return Service Details
1.6: Delivery Information
1.6A: Not Available for Delivery
1.5: Return Verdict (can provide service?)
1.7: Display Information
<<user interface>>
: VolunteerService Interface
<<state dependent control>>
: MainProcessing
1: Request Food Need
1.1: Send User Request
1.4: Contact Volunteer
: Food Beneficiary
: FoodDelivery Volunteer
2: Check Availability
2.1: Return Available Supplies Information
<<external I/O device>>
: DonationCentre
<<I/O device interface>>
: DonationCentre Interface
3.1: Send Inventory Information
1.2: Check Availability
1.3: Return Available Food Items
3: Send Food Provisions
<<subsystem>>
: FoodInventory
: Food WealthGiver

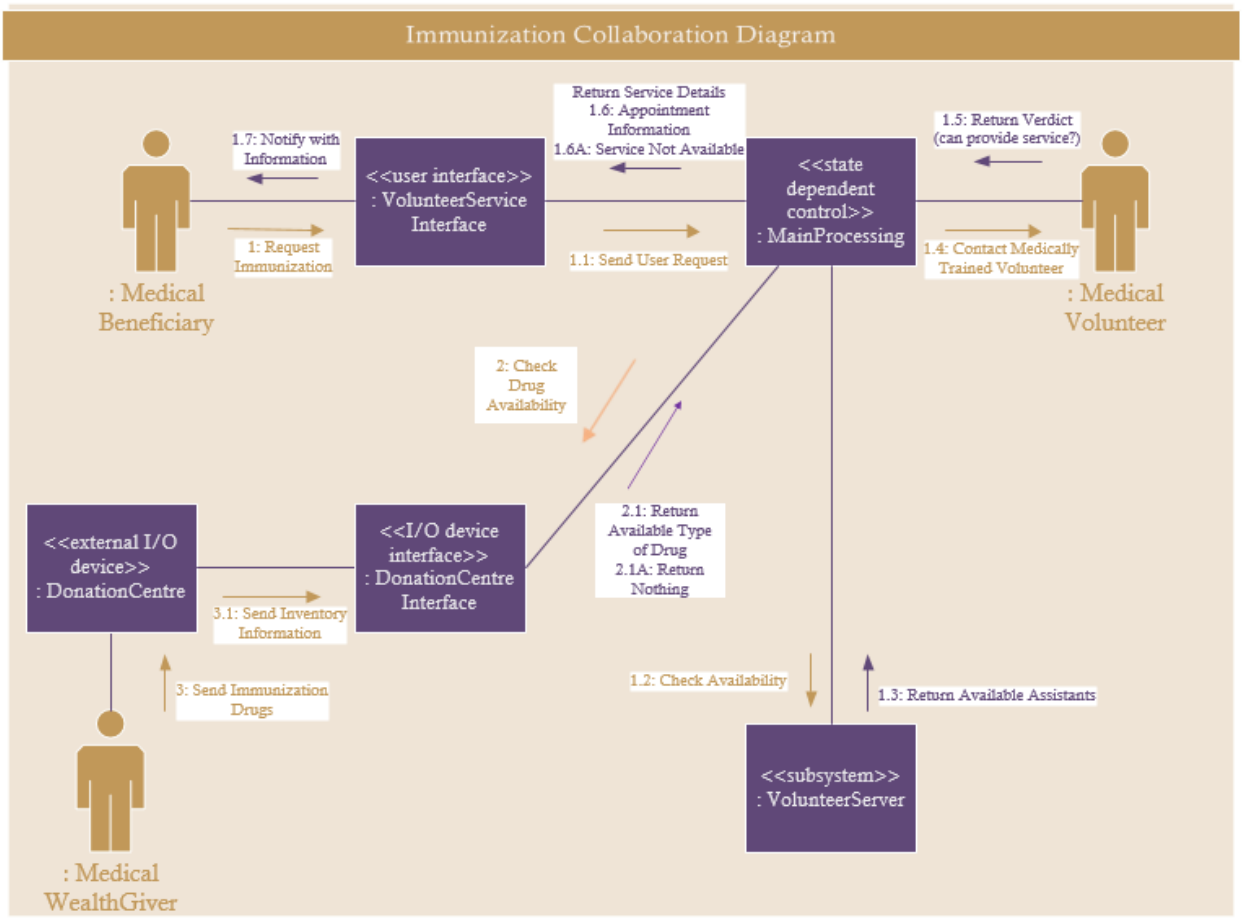

Child Care Collaboration Diagram
Return Service Details
1.6: Meeting Information
1.6A: Service Not Available
1.5: Return Verdict (can provide service?)
1.7: Display Information
<<user interface>>
: VolunteerService Interface
<<state dependent control>>
: MainProcessing
1: Request Babysitting
1.1: Send User Request
1.4: Contact Babysitter
: ParentGuardian Beneficiary
: ChildCare Volunteer
2: Check Availability
2.1: Return Available Supplies Information
<<external I/O device>>
: DonationCentre
<<I/O device interface>>
: DonationCentre Interface
3.1: Send Inventory Information
1.2: Check Availability
1.3: Return Available Assistants
3: Send Child Care Supplies
<<subsystem>>
: VolunteerServer
: WealthGiver


Hygiene Care Collaboration Diagram
: Hygiene Beneficiary
1.7: Notify with Information
1: Request Hygiene Care
<<user interface>> : VolunteerService Interface
Return Service Details
1.6: Appointment Information
1.6A: Service Not Available
1.1: Send User Request
<<state dependent control>> : MainProcessing
1.5: Return Verdict (can provide service?)
1.4: Contact Volunteer
: Hygiene Volunteer
2: Check Availability
2.1: Return Available Supplies Information
<<external I/O device>> : DonationCentre
3.1: Send Inventory Information
<<I/O device interface>> : DonationCentre Interface
3: Send Hygiene Supplies
1.2: Check Availability
1.3: Return Available Assistants
: Hygiene WealthGiver
<<subsystem>> : VolunteerServer

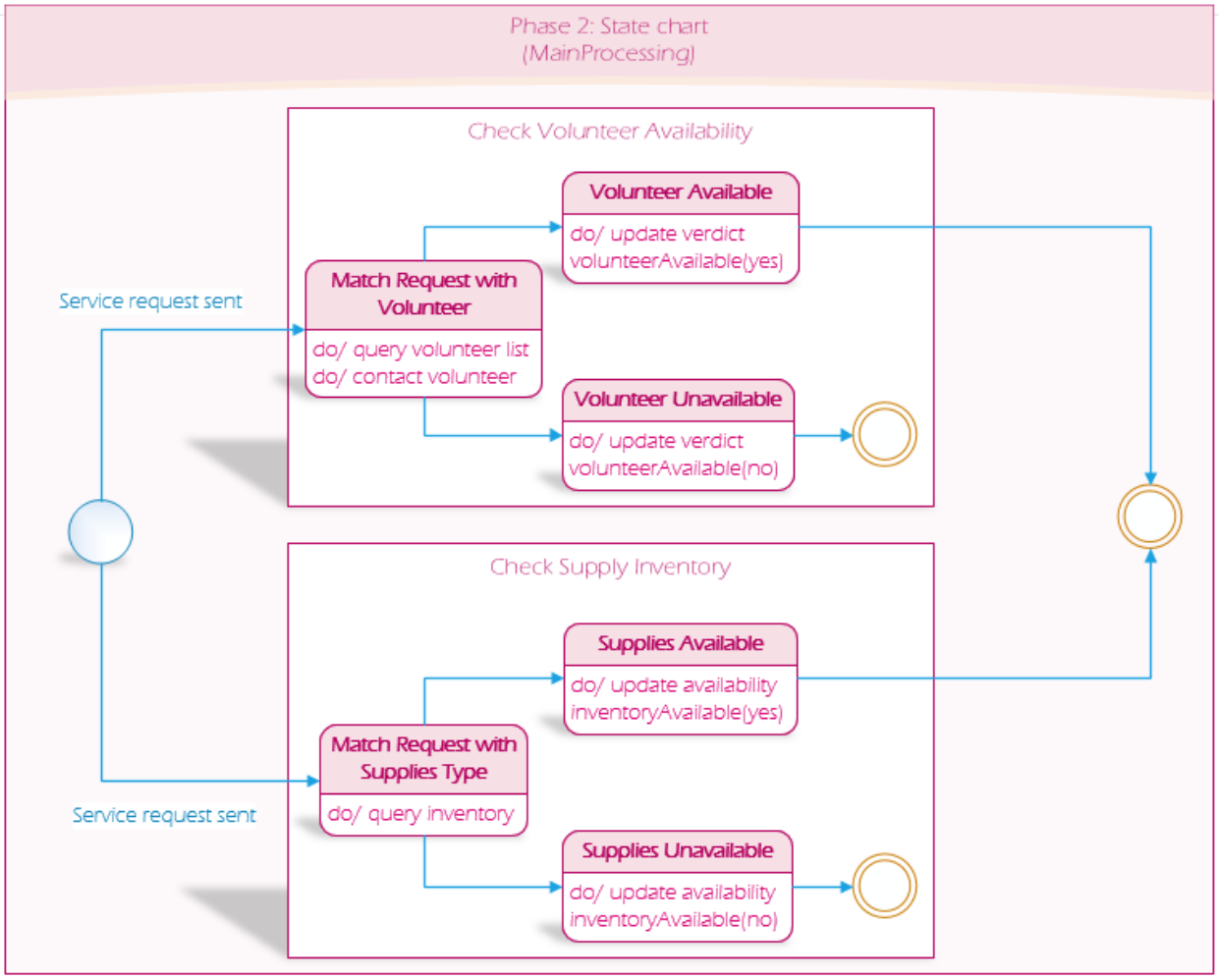

Haircut Collaboration Diagram
: Hygiene Beneficiary
1.7: Notify with Information
1: Request Hygiene Care
<<user interface>> : VolunteerService Interface
Return Service Details
1.6: Appointment Information
1.6A: Service Not Available
1.1: Send User Request
<<state dependent control>> : MainProcessing
1.5: Return Verdict (can provide service?)
1.4: Contact Barber
: Barber Volunteer
2: Check Availability
2.1: Return Available Equipment Information
<<external I/O device>> : DonationCentre
3.1: Send Inventory Information
<<I/O device interface>> : DonationCentre Interface
3: Send Haircutting Equipment
1.2: Check Availability
1.3: Return Available Assistants
: Hygiene WealthGiver
<<subsystem>> : VolunteerServer


Health Care Collaboration Diagram
Return Service Details
1.6: Appointment Information
1.6A: Medical Report Information
1.6B: Service Not Available
1.7: Notify with Information
<<user interface>>
: VolunteerService Interface
<<state dependent control>>
: MainProcessing
1.5: Return Verdict (can provide service?)
1: Request Medical Check-Up
1.1: Send User Request
1.4: Contact Medically Trained Volunteer
: Medical Beneficiary
: Medical Volunteer
2: Check Availability
<<external I/O device>>
: DonationCentre
<<I/O device interface>>
: DonationCentre Interface
2.1: Return Available Supplies Information
3.1: Send Inventory Information
1.2: Check Availability
1.3: Return Available Assistants
3: Send Health Care Supplies
<<subsystem>>
: VolunteerServer
: Medical WealthGiver


Blood Donation Collaboration Diagram
Return Service Details
1.6: Appointment Information
1.6A: Service Not Available
1.7: Notify with Information
<<user interface>>
: VolunteerService Interface
<<state dependent control>>
: MainProcessing
1.5: Return Verdict (can provide service?)
1: Request Blood
1.1: Send User Request
1.4: Contact Medically Trained Volunteer
: Medical Beneficiary
: Medical Volunteer
2: Check Blood Availability (Blood Type Congruent?)
<<external I/O device>>
: DonationCentre
<<I/O device interface>>
: DonationCentre Interface
2.1: Return Available Type of Blood
2.1A: Return Nothing
3.1: Send Inventory Information
1.2: Check Availability
1.3: Return Available Assistants
3: Send Blood Donation
<<subsystem>>
: DonationCentre
: Medical WealthGiver


Immunization Collaboration Diagram
: Medical Beneficiary
1.7: Notify with Information
1: Request Immunization
<<user interface>> : VolunteerService Interface
Return Service Details
1.6: Appointment Information
1.6A: Service Not Available
1.1: Send User Request
<<state dependent control>> : MainProcessing
1.5: Return Verdict (can provide service?)
1.4: Contact Medically Trained Volunteer
: Medical Volunteer
2: Check Drug Availability
2.1: Return Available Type of Drug
2.1A: Return Nothing
<<external I/O device>> : DonationCentre
3.1: Send Inventory Information
<<I/O device interface>> : DonationCentre Interface
3: Send Immunization Drugs
: Medical WealthGiver
1.2: Check Availability
1.3: Return Available Assistants
<<subsystem>> : VolunteerServer

Part 3: State chart



Part 4: Implementation of a Use Case

```
// include package(s)
package iVolunteer;

/*
 * General delivery service that specific services must implement.
 */
public abstract class DeliverService {
    /*
     * Interact with user to
     * send and receive their request.
     */
    abstract class VolunteerServiceInterface {
        /*
         * Handle sending and receiving requests from/to the beneficiary.
         */
        abstract void notifyOfNeed(String serviceType);
    }

    /*
     * Centre of processing sent/received messages
     * between various systems/interfaces.
     */
    abstract class MainProcessing {
        /*
         * Specify information such as: appointment information,
         * whether the service can/cannot be provided,
         * and processing user requests.
         */
        abstract void serviceDetails(boolean verdict, String serviceType);
    }

    /*
     * Manage requests for the donation centre.
     */
    abstract class DonationCentreInterface {
        /*
         * Process inventory information
         * for items sent to the donation centre
         * from the wealth giver.
         */
        abstract boolean inventoryAvailable(String serviceType);
    }

    /*
     * Manage all the volunteers apart of the iVolunteer system.
     */
    abstract class VolunteerServer {
        /*
         * Process available assistants who fit the beneficiary request.
         */
        abstract boolean volunteerAvailable(String serviceType);
    }
}
```

Part 5: Activity Diagram