

SOFTWARE ENGINEERING



REPORT 3

Automatic Delivery System

Li Liu
Bowen Pan
Tuo Wang
Qiannan Li
Mehmet Aktas
Marc Gamell

Home [web page](#)
Code [web page](#)

Fall 2012

Contents

Contribution breakdown	1
Summary of Changes	2
1. Customer Statement of Requirements	4
1.1. Problem	4
1.2. Automatic delivery system	5
1.2.1. Simplified system-user interaction	6
1.2.2. Web and mobile interface	6
1.2.3. Multiple Delivery	6
1.2.4. Database of addresses and passwords	7
1.2.4.1. Barcode scanner	7
1.2.5. Automatic battery charging	7
1.2.6. Location Capability	7
1.2.7. Security concerns	8
1.2.8. Delivery Confirmation	8
1.2.9. Automatic failure detection - Maintenance	8
1.3. Scalability of our solution	9
1.3.1. Delivery delay	9
1.3.2. Mutiple-floor deliveries	9
1.3.3. Multiple-building deliveries	9
2. Glossary of Terms	10
3. System Requirements	13
3.1. Enumerated functional user stories	13
3.2. Enumerated non-functional user stories	15
3.3. Acceptance tests for user stories	15
3.4. On-screen Appearance Requirements	21
4. Functional requirements specification	25
4.1. Architectural style	25
4.2. Stakeholders	26
4.3. Actors and goals	26
4.4. Use case diagram	26
4.5. Use cases casual description	28
4.6. Traceability matrix	29
4.7. Use cases fully-dressed description	29
4.7.1. BookDelivery	29
4.7.2. MoveRobotToPoint	31
4.7.3. PositionInspection	33
4.7.4. SetMotorSpeedL/SetMotorSpeedR	33
4.7.5. ChargeBattery	34
4.7.6. Login	34
4.7.7. Access Delivery Record	35
4.7.8. Notify Error	36
4.7.9. Track Delivery Status	36
4.7.10. Authentication to Robot	37
4.7.11. Notify Receiver	38

4.7.12. Obtain Help	38
4.7.13. Pickup Package	39
4.7.14. Delivery	39
4.7.15. ManageUser	41
4.7.16. EditUserInformation	41
4.8. Acceptance tests for Use Cases	42
4.9. System sequence diagrams	59
4.10. Risk management	59
5. Effort Estimation Using Use Case Points	69
6. Domain Analysis	71
6.1. Domain Model	71
6.1.1. Concept definitions	71
6.1.1.1. Boundary concepts	71
6.1.1.2. Internal concepts	72
6.1.1.3. Summary of concepts	74
6.1.2. Attribute definitions	74
6.1.3. Association definitions	76
6.1.4. Traceability matrix	80
6.2. System Operation Contracts	80
6.3. Mathematical Model	87
6.3.1. Track detection	87
6.3.2. Path resolution	87
7. Interaction Diagrams	94
7.1. Login and BookDelivery	94
7.2. PickUpPackage and Delivery	96
8. Class Diagram and Interface Specification	101
8.1. Class Diagram	101
8.2. Data Types and Operation Signatures	103
8.3. Traceability matrix	111
8.4. Design Patterns	111
8.4.1. Singleton Pattern	111
8.4.2. State Pattern	112
8.5. Object Constraint Language (OCL)	118
9. System Architecture and System Design	123
9.1. Architectural Styles and Subsystems	123
9.1.1. Client/Server and Master/Slave Architectures	123
9.1.2. Three-tier architecture	124
9.1.3. Facade Pattern	125
9.1.4. Event-driven Architecture	125
9.1.5. Summary	125
9.2. Subsystems	126
9.3. Mapping subsystems to hardware	126
9.4. Persistent Data Storage	127
9.5. Network Protocol	127
9.5.1. Network between Server and Clients	130
9.5.2. Network between Server and Robot	130
9.6. Global Control Flow	130
9.6.1. Execution Orderness	130
9.6.2. Time Dependency	131
9.6.3. Concurrency	131
9.7. Hardware and Software Requirements	131

10. Algorithms and Data structures	133
10.1. Algorithms	133
10.2. Data structures	135
11. User Interface Design & Implementation	138
11.1. User Interface Design & Effort Estimation	138
11.2. Interaction of User Interfaces	138
11.3. Detailed User Interface Implementation	146
11.3.1. Register and Edit information	146
11.3.2. Book Delivery	147
11.3.3. Track Deliveries Status	147
11.4. Improvement in UI Implementation Compared with UI Design	149
12. Design of Tests	150
12.1. Test cases in Unit Testing	150
12.1.1. Strategy to Do Unit Testing	150
12.1.2. Test Cases in Unit Test	150
12.2. Test Coverage	151
12.3. Integration Test strategies	151
13. History of Work, Current Status, and Future Work	153
13.1. History of Work	153
13.2. Current Status	154
13.2.1. Communication Mechanism among software components	154
13.2.2. Features in Auto Delivery System	155
13.2.3. Non-Software Problems and Solutions	158
13.3. Future Work	160
Bibliography	162
A. Existing commercial solutions	165
A.1. Traditional system	165
A.2. Integrated delivery system using pipes	165
A.3. Drug delivery using robots	165
A.4. Automated warehouses	166
B. Interaction Diagram for UC-8 Delivery	167

CONTRIBUTION BREAKDOWN

Responsibility	Li Liu	Bowen Pan	Tuo Wang	Qiannan Li	Mehmet Aktas	Marc Gamell
Chapter 1	5	10	20	20	10	45
Chapter 2			33	33		33
Chapter 3	10	10	20	20	15	25
Chapter 4	12	12	12	12	12	40
Chapter 5	16	16	16	16	16	16
Chapter 6	10	10	10	10	10	50
Chapter 7. UML diagrams					50	50
Chapter 7. Prose descr. of diagram					50	50
Appendix 2. Alternative solution description					50	50
Chapter 8. Class diagram and description	16	16	16	16	16	16
Chapter 8. Signatures	16	16	16	16	16	16
Chapter 9. Styles					50	50
Chapter 9. Package diagram					50	50
Chapter 9. Map hardware					50	50
Chapter 9. Database					50	50
Chapter 9. Other design aspects					50	50
Chapter 10. Algorithms and data structures	33		33	33		
Chapter 11. UI appearance		100				
Chapter 11. UI description		100				
Chapter 12. Testing design			50	50		
Chapter 13.		25	25	25	12	12
Project Management						100

Table 0.1.: Contribution Breakdown for report 3

Responsibility	Li Liu	Bowen Pan	Tuo Wang	Qiannan Li	Mehmet Aktas	Marc Gamell
program coding					50	50
unit testing			50	50		
Integration test			50	50		
Debugging					50	50
Design and Maintain Database					50	50
Data Collection					50	50
Brochure/Flyer Preparation		100				
Slides Preparation					50	50
Program Documentation	25	40	17.5	17.5		
project Management	16.6	16.6	16.6	16.6	16.6	16.6

Table 0.2.: Contribution Breakdown for demo 1

Responsibility	Li Liu	Bowen Pan	Tuo Wang	Qiannan Li	Mehmet Aktas	Marc Gamell
Demo 2		10	10	10	35	35

Table 0.3.: Contribution Breakdown for demo 2

SUMMARY OF CHANGES

Following are the brief summary of changes in our project:

- In the website design, we update the website(including the new website address and the layout) and upload the newest information.
- Add Bibliography into Table of Contents
- In Section 1.2.6 Location Capability, we add more explanations about how the tapes helps in guiding the moving way.
- In Chapter 2 Glossary of Terms, we modify the definitions of users and administrators.
- In Section 3.3 Acceptance tests for user stories, we add the alternative case for ACT1 and modify the following explanations.
- In Chapter 7 Interaction Diagram, we made the following changes:
 - In Section 7.1, Login and BookDelivery, we change expressions in explaining the interaction diagram 'Logn and BookDelivery'
 - In Section 7.2, PickUpPackage and Delivery, we modified the description of 5 interaction diarams in order to better illustrate how these 5 interaction diagrams achieve the use case PickUpPackage and Delivery together. We add explanations to the initiation factor of these two use cases, and illustrate how we come up with the PickUpPackage/Delivery algorithm that can handle with the time-unpredictable circumstance
- In Chapter 8 Class Diagram and Interface Specification, we made the following changes:
 - In Section 8.1, Class Diagram, we refined the description of each packages and the relationship between each other.
 - In section 8.2, Data Types and Operation Signatures, we change all 'partial class diagrams'. We add classes NonBookedDeliveryException, ServerInitializedException, ServerNonInitializedException and SystemStatus into the partial class diagram 'Package Logic'; We also add classes TimeoutException into partial class diagram 'package resources.communication'; we also added the description of the resource. datacontroller package.
 - We added the Design Patterns in Section 8.4
- In Chapter 9 System Architecture and System Design, we revise description of some architectures and some system design pattern to make it more readable for reader to understand.
 - In Section 9.1, we add explanations to Facade Pattern and Event-driven Architecture, and how our system implement these patterns.
 - In Section 9.4, Persistent Data Storage, we add explanations to how all persistent classes can be linked together to retrieve full information of a delivery.
 - In Section 9.5.2, we use sockets as interface to communicate between robot and server, instead of HTTP protocol.
 - In Section 9.6.3, we add explanations and reference regarding multi-threading using Java RMI
- In Chapter 10 Algorithms and Data structures, we added more detailed descriptions to make this part more clear.
 - In Section 10.2 Data Structures, we add descriptions and plots explaining how hash tables and linked list can be used into our system.
- In Chapter 11 User Interface Design & Implementation, we re-organize the structure of this chapter to make it more readable. We also modified description of actual UI implementation and description of improvements compared with mock-up design.

- In Chapter 13, we describe the history work, current status (including detailed explanation to the features of robot and server) and future work.

1. CUSTOMER STATEMENT OF REQUIREMENTS

Communication plays an important part in human society and has been always a great challenge to human being. The fast development of computer network such as Internet provides an incompatible convenient access and powerful tool for people to get and share the information they need and so we are unprecedentedly linked with each other in this Information Age. The recent electronic version of information does tremendously facilitate our communication, but it is not sufficient and may not be convenient enough in some cases.

As Kevin Ashton put it in 1999:

We're physical, and so is our environment ... You can't eat bits, burn them to stay warm or put them in your gas tank. Ideas and information are important, but things matter much more. Yet today's information technology is so dependent on data originated by people that our computers know more about ideas than things.

Integrating physical objects into the information network and letting them become active participants in the delivery processes may be the next stage of the communication development. In that sense, our project comes into stage to help people handle some inadequacies of the Internet and partly implement the idea of **Internet of Things** (see [12] and [55]).

1.1. Problem

Typically, in an office building there exists some necessities in terms of transferring packages and documents, such as:

1. Postman needs to distribute some correspondence to some rooms or a postbox in a daily basis. Therefore, he or she may need to dispatch this work to another person or the final recipient may need to check the postbox regularly.
2. Moreover, in some cases, the correspondence is delivered to a wrong person. This introduces unnecessary delays and worries to both the recipient and the postman/post office.
3. Some collaborators in different offices (sometimes in different floors) need to share documents between them.
4. Some big file transfers done by regular e-mail or by using a central server and the building network lasts 2 or 3 hours¹.
5. For security reasons, some people do not trust network transfers (even when encrypted) for confidential files. That is why some institutions still prefer internal regular mailing system.
6. Usually, departments have only a centralized powerful printer, which all the members in the department can share. Therefore, they may need to take a walk to the printing room for maybe a single paper.
7. Similarly to the printer scenario, in huge buildings there are food courts in which people might like the delivery service.

¹40GB file transfer in a perfect 100Mbps network would last about an hour

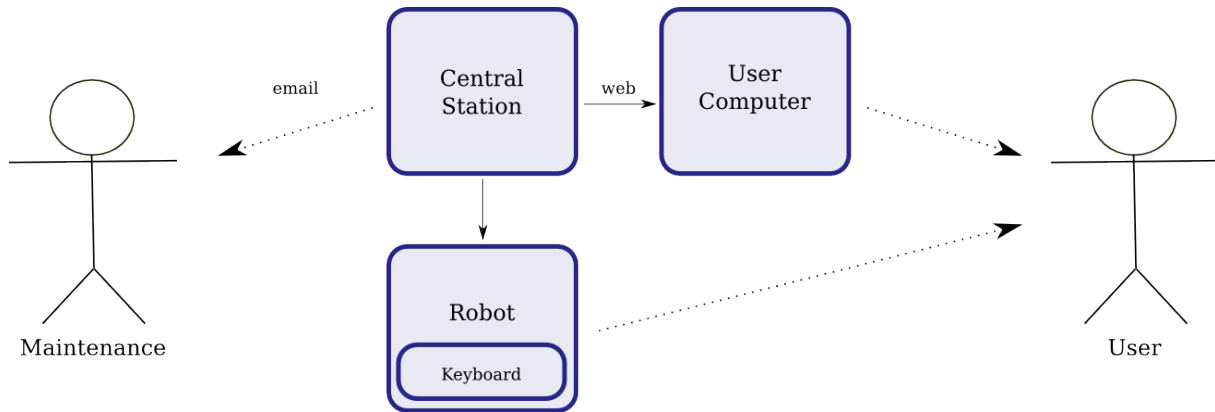


Figure 1.1.: Structure of the system.

The same delivery problem can be seen in many other environments. The need of distributing drugs to all the patients in a hospital can be given as another typical delivery example. There is no need to say how critical this process is, and how a machine can be used to improve reliability in front of human mistakes.

The problem that all of these processes or situations have in common is that all are **time-consuming** and, therefore, potentially decrease the work efficiency of employees. Moreover, the departments can feel forced to hire some workers to deal with this kind of service.

1.2. Automatic delivery system

We propose a system that is going to deal with all the situations above! Briefly, our system will enable users to book delivery services, will locate positions of both the sender and recipient. Eventually, our system will pick up the object from the sender and will deliver it to its recipient. All these things will be performed in an automated and secured manner.

We propose to deploy a user-targeted robot aimed to help the correspondence sharing and distribution. The robot will have moving capability and will be able to hold multiple little packages (it will contain a box with multiple sections). We will provide convenient ways to users to book a delivery such as web or mobile interface.

Our system can make user's lives much more time-efficient and, at the same time, will provide the department a cost-effective, easy-maintainable solution. Moreover, the system will be scalable in several aspects, as discussed in detail in section 1.3.

In the next sections you will find how our system will make your life easier! But for the impatient, the most important features from the user point-of-view are the following:

1. Web and Mobile Interface
2. Multiple Delivery (secure sectioned storage with a locked lid in every section)
3. User Information Storage
4. Barcode Scanner
5. Automatic Battery Charging
6. Security Guarantees
7. Automatic Failure Detection
8. Detection of routes via marks on floor and magnets on important points

1.2.1. Simplified system-user interaction

As long as the main advantage to the user is to gain efficiency, the points of interaction between the system and the user will be minimized. The compulsory contact points will be, however:

- The user interface (web and mobile application): to book a delivery pick up or to track a delivery in progress.
- The keyboard integrated in the robot: to confirm the identity of the sender/receiver.
- The e-mail: to receive notifications such as soon pick up/delivery or possible problems.
- The ‘multi-sectioned’ security box on top of the robot: to deposit or take off the packages/mails.

Another important point to take in mind is the system maintenance. Although our robot will be designed to be as much autonomous as possible, some situations can’t be avoided without human intervention. That is why the costumer will need to allocate resources for these purposes.

Note that this design helps accomplishing the following goals:

- There is no need to give a training for the final users, because the system will be very intuitive.
- The maintenance team may require some training sessions, just to ensure that they have a clear picture of the system, which information is given in case of an error, and what the maintenance team is expected to do in case of a given error.

All the details about these contact points or other features that may ease the user experience are discussed in the following sections.

1.2.2. Web and mobile interface

Our robot delivery system innovatively allows users to request or to book a delivery service through the computer [56] in front of their desks or a convenient Android (see [1]) application on their phones (see [19]).

Our human-robot interaction interfaces will be simpler and more interactive than other systems (see a detailed comparison of our solution with other systems in section A.2, when talking about the user interface). Users will get easy access to the system with the user-friendly GUI on the computer or smartphones. Through this control platform, the users can just click and book a detailed request, and eventually the robot will appear in front of user’s office to pick up the packages (see [2]).

1.2.3. Multiple Delivery

The auto delivery system should be able to deliver multiple packages [17] at the same time. These packages might be issued by the same person or by different people, to a single address or different addresses. This feature brings about two new questions. One is, how can we ensure the recipients of these packages receive their corresponding packages without being mixed up and getting the wrong ones? Another question is, how to prevent bad things happen, for example, some recipient take their own package, as well as taking someone else’s on purpose.

Automatic Delivery System is able to deal with this! The container of the robot is divided into several partitions, each with a slide and an automatic lock to cover the partition. When a delivery task is assigned to the system, it generates a six-digit password and send it to the recipient by email or messages. The recipient needs to enter this six-digit password (or its own username and password) using the keyboard of the robot. If the system finds this correct, the corresponding cover will be unlocked and the recipient will be able to access only to his or her package. In this way, different deliveries can be separated physically, and the recipient will need to be authenticated before accessing its package.

Of course, this multiple delivery issue arises the question of package capacity. Actually the package capacity will be determinant factor for the maximum number of packages that can be delivered simultaneously but the users that cannot be served during the time that the system is fully loaded can then be scheduled as the prior users whom will be served right after the current task.

In short, the philosophy is here to provide a first come first serve basis delivery service which can be used for multiple users and packages as far as the capacity allows.

1.2.4. Database of addresses and passwords

As our system is user-centered. Therefore, the sender user shall do delivery requests **based on recipient names**, not based on places. This means that the user only needs to worry about who is the recipient of the document or package he or she wants to share. The user does not need to worry about *where* it must be delivered any more!

To allow this important feature, the customer may need to allocate a responsible for maintaining the databases. However, to reduce the impact on maintenance costs for our system, the final user shall be able to modify its own information (password, name, real address) from any of the convenient interfaces.

1.2.4.1. Barcode scanner

This system uses a barcode scanner as a input and package authentication [57] to the robot. A user will feel to be of great convenience if he or she has more than one item to be dispatched, which all of them have barcode. When the user scans the barcodes of some item, the robot will receive corresponding package information (including destination address of each package) stored in the remote database (see [7]).

After the user put every package into the robot into every partition of the container, and close each cover. it will leave and dispatch each of them separately. This guarantees the comfort of the user, as he or she does not need to worry about dispatching all these packages entering each destination individually.

1.2.5. Automatic battery charging

The customer nor the user doesn't need to worry about the charging of the batteries. The system will handle this hazard.

When the robot is in low battery situation, the delivery robot can automatically drive to the designated charging area. The robot charging device will connect the ground charging device automatically and begin the charging process. When this process is finished, the robot will then automatically go back to the original area to wait for user's request. Also, the system will be charged using the inductive method, which realizes the energy transfer through unconnected coupling mode based on electromagnetic induction.

As the user will see, our robot aims to realize the automation, intelligence and free of human maintenance during the whole charging process (see [60]).

1.2.6. Location Capability

The major function of this robot is that user can use this system to transfer some items from an original address (derivated from the sender's name, e.g. Room No. 101) to the destination address (derivated from the recipient's name, e.g. Room No. 105). With the capacity of location, including moving ability and positioning capability, the user can easily assign the robot to come to current location, pickup items, and heads to designated address without moving a single step!

The robot moves with a controller, an engine and its four wheels. It has a train-like moving track, which consists one white tape and one black tape lying on the ground (which can be installed easily). We install the photosensitive sensors in front of the robot. It can sense the reflection of light from the ground and show the resistor differences between the white and black tapes. By detecting the combination of light (reflected from the white and black tape) on the floor, the robot can move in a pre-set moving track, without worrying about got the wrong way, or hit the wall etc.

Meanwhile, whenever the robot is assigned origin and destination addresses by any means (webpage, barcode scanner or Android app), it is available to detect its current distances from current address to the original address, or from original address to the destination address, through magnet sensors embedded on it. In this way, robot will be able to move and go to the address he is supposed to reach.

1.2.7. Security concerns

As information is important to users, they shall be confident that some principles of information security are accomplished. Typically, a user may want (see [11]):

1. Confidentiality: prevent the disclosure of information to unauthorized recipients. A given information won't be accessed by anyone who is not the recipient.
2. Authenticity and Integrity: the information will arrive as the sender sent it: without external modifications
3. Non-repudiation: the final recipient will receive the package, and he can't deny he or she haven't.

The environment will be recorded with a camera in order to keep a record of possible malicious actions. On the other hand, the robot will provide the recipient with a convenient interface (a keyboard), where he or she needs to type its unique and secret credentials. Only when this is done a green light will indicate which section or compartment of the box shall he or she open.

The user can see the previous security concerns applied to our system. Sometimes, however, ease-of-usage is putted aside prioritizing security:

1. The recipients shall type its credentials in order to be granted to access its packages
2. As the only persons that can access the packages are the sender and the recipient, and during the delivery process, the package is situated in a box that will never be opened, the property of authenticity and integrity are guaranteed.
3. when typing the password and the 'Confirm' button, the recipient is guaranting that he or she is accessing the package. Therefore, the property of non-repudiation is accomplished.

1.2.8. Delivery Confirmation

Now that the robot will be able to reach your office with the mail/package to be delivered, the user will be able to pick up the mail/package. The user needs to **confirm** that he or she received the package. The simpler way to confirm this is to press the 'Confirm' button embedded on the robot, much the way as you signed for a package to a postman. After one confirm, the robot will leave as if the postman will leave after delivering your package.

In the situation that the recipient is not available or not present at the office, the robot will leave an electronic message to the recipient, wait for some time at the door (tentatively 2 min), then leave to the next recipient if still nobody picked up the package. The robot will come to this place for up to three times, and then give package back to the sender.

This is implemented by updating confirmation information in the database in time after entering the password correctly and pressing the 'Confirm' key, as well as by setting up a timing scheme in case nobody answers.

1.2.9. Automatic failure detection - Maintenance

However, there are some situations that even the most robust systems can't avoid. That is why our system, besides being robust will be fault-tolerant. Some possible situations are:

- It may happen that the robot gets lost (both lose its path or, although being in the path can't find a certain location). If this happens, in most cases the customer won't need to worry because our robot will be equipped with path-recovery algorithms. However, if after a predefined amount of time (e.g. 5 minutes) the robot does not localize itself, a message will be sent to the maintenance team with the last-known location of the robot.
- Also, it may happen that the robot runs out of battery unexpectedly or some subsystem (such as the communication module) simply stops working. Note that, in this case the robot won't be able to send any message to the rest of the system to ask for help. However, this is not a problem because the robot is always in contact with the central station. In case the central station can't contact the robot in a certain amount of time an error will be reported to the maintenance team.

Therefore, the customer does not need to worry about the malfunction of the system, because if something goes wrong, the maintenance team will receive an e-mail.

1.3. Scalability of our solution

As introduced in chapter 1.2, our system is scalable in several aspects. In this section we will discuss what do we mean with scalability and will give some hints about how future projects can handle it.

1.3.1. Delivery delay

On the one hand, our system is scalable in terms of the delivery delay. In case our system receives, in a given period of time, more requests than which can handle, the senders will experience delays. This problem can easily become a bottleneck. However, if the customer finds that the delivery delay is too high, it can simply add more robots to the network. Our system allows this feature, because there is no problem (but the cost) of adding more than one robot.

1.3.2. Multiple-floor deliveries

On the other hand, in the problem description (chapter 1.1) we introduced the fact that sometimes deliveries must be done in different floors. However, in our approach we proposed a solution that is only able to deal with single floors.

To deal with the multi-floor problem our system could use the elevator that many buildings have. For this purpose, the customer only needs to provide the project team an API² to request the elevator to a certain floor, to choose the destination floor and to request the current floor of the elevator.

1.3.3. Multiple-building deliveries

Finally, our project can still scale to deal with intra-building deliveries. However, this requires much more caution:

1. Location via path on the floor is not valid for outdoor environment. This can be solved by placing a GPS in the robot and having two location systems: when the robot is outdoors, the routes can be described in a map, which the robot can follow via GPS; when the robot is indoors, where the GPS signal sometimes is very low, the robot can be guided using the path on the floor.
2. As the floor on the outside is typically heterogeneous and have irregularities, the robot must include much more robust wheels and motors than an indoor-only robot.

²We will try to get this API for the testing building. As long as we think that the elevator provider will not share any of its APIs, we won't be able to test this feature.

2. GLOSSARY OF TERMS

The roles of our system are:

Customer The enterprise or department that might want to deploy our system

User Individuals which will actually use the system (the ‘senders’ and the ‘recipients’ of packages)

Administrator Individuals which will actually maintain the system operation and manage user information.

Other terms that may need clarification are (ordered alphabetically):

Arduino Uno It is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. More information can be found in [5].

Arduino Motor controller shield It is a shield based on the L298, which is a dual full-bridge driver designed to drive inductive loads such as relays, solenoids, DC and stepping motors. It lets one drive two DC motors with one’s Arduino board, controlling the speed and direction of each one independently. One can also measure the motor current absorption of each motor, among other features. The shield is TinkerKit compatible, which means one can quickly create projects by plugging TinkerKit modules to the board. More information can be found in [4].

Arduino WiFi shield It is a shield enables Wi-Fi connectivity with an onboard PCB antenna from ZeroG Wireless. It provides 802.11b connectivity and is a direct drop-on plug-and-play solution to one’s Arduino Uno.

Automatic delivery system A transport system that allows the senders to deliver mails or packages to receivers with a more efficient way, without the inconvenience associated with delivering goods manually. In this automatic delivery system, a sender can book a delivery by visiting its webpage. A robot will come to the sender’s place to pick up the mail/package and send it to corresponding receiver without moving a step. The receiver also do not need to move a step to get his/her package from the robot.

Barcode A barcode [24] is an optical machine-readable representation of data related to the object to which it is attached. Barcodes represents data by varying the space and width of parallel lines, or 2D patterns like rectangles and dots. In automatic delivery sytem, a barcode may be attached to a mail or package after it’s corresponding delivery information(sender/receiver’s address, delivery password) has been stored in the database. After the barcode scanner scanned the barcode, the robot can move to the receiver’s address for a mail/package delivery.

Barcode scanner A barcode scanner [25] is an electronic device for reading printed in the surface of mails or packages. Like a flatbed scanner, it consists of a light source, a lens and a light sensor translating optical impulses into electrical ones. A barcode scanner can be used as an input to the automatic delivery system to book a delivery.

Central station A general management unit for the automatic delivery system. It is responsible for interconnecting with robots (including analysing moving paths, receiving input information from robot etc.), managing webpages, monitoring each delivery cases, and handling with exceptions.

Confirm button A button embedded in the robot to confirm actions, such as the end of password typing or the end of a pick up or delivery process.

Database A database is a collection of information that is organized so that it can easily be accessed, managed, and updated. In one view, databases can be classified according to types of content: bibliographic, full-text, numeric, and images. More information about database can be found in

[26]. In our system, the database will manage information about users or delivery history, for example.

Delivery This term can refer (i) to the process of extraction of the package from the box by the receiver or (ii) to the whole process of delivery (from the booking to the actual delivery).

Delivery booking The main interaction between the user and the system occurs when it requests a delivery via any of the available interfaces (web or mobile). Thanks to this booking, the robot will appear in front of the sender's office to pick up the package.

Delivery notification When a package or mail is going to be sent to the receiver, a message similar to "You have a package that will be delivered soon" will be sent to the receiver several minutes before the robot arrives at his or her office. This message is a delivery notification and it is used so that the receiver can get the notice in advance and get prepared for receiving the package or mail.

Delivery service A delivery service refers to a transport service organised by a supplier or a sender to take goods to receivers.

GPS Global Positioning System [31] is a satellite-based system that allows a device on earth to know its position (with a certain precision).

Inter-building delivery A delivery process where both the sender and the receiver are in different buildings.

Intra-building delivery A delivery process where both the sender and the receiver are in the same building (may or may not be in the same floor).

Keyboard A set of numeric keys embedded to the robot that enables users to enter numbers as "password" to check the authenticity of the user.

Last-known location of the robot If the robot gets lost, this term refers to the last point (e.g. office CoRE 623) in which the robot maintained communication with the central server.

Location capability The ability of the robot to know the important points on the building (such as the offices) and to know how to move from one point to another. This is achieved in our first approach to the system via location marks.

Location marks Location marks refers to a track laying on the aisle and several magnet chips attached on the wall in the office building. The track is made up of a line of white tape, which the robot moves along the white line, and a line of black tape that prevents the robot from getting interfered by other objects on the road. Magnet chips are used to locate each room by working with sensors in the robot.

Mobile interface A web-based user interface working on Android smart phones, where users can view and execute operations through this application. These operations include: book a delivery, check status of package/mail, personal information management, and system maintenance.

Mobile robot A mechanical device that can receive instructions about moving to different locations in a building, carry multiple mail/packages in a secure manner and deliver packages automatically.

Multiple delivery Our system allows multiple delivery. This means that the customer will need to invest less in robots and the user will experience much less delay in the pick up or delivery process.

Multiple-floor delivery A delivery process where both the sender and the receiver are in different floors.

Package/Letter The correspondence that wants to be transported from one point to another one.

Pick up This process begins when the robot appears in front of the sender's office. It contains several steps, such as the sender authentication or the introduction of the correspondence in the robot's box.

Properties of security Refers to the confidentiality, authenticity, integrity and non-repudiation properties of a secure information transfer. See [11] for details.

Sectioned secure-box A delivery box containing several sectioned parts on top of the robot. The sectioned box aims to provide multiple packages or mails delivery to different receivers at the same time, while guarding the safety of each package/mail by connecting the open/close of each section with a password set by the sender.

System (see Automatic delivery system) The entire set composed of the mobile robot, the central controller or server, the user interface and the location marks.

System exception System exception refers to errors that happened when the system is running. In Automatic Delivery System, system exception includes, but is not limited to:

- Robot out of track;
- Robot out of power;
- Disconnection between robot and central station;
- System unexpected reboot;
- System unexpected shut down;
- Robot missing target location;

Use case A list of steps, typically defining interactions between a role (known in UML as an ‘actor’) and a system, to achieve a goal. The actor can be a human or an external system.

User address A sender’s/receiver’s address refers to the room location of the package/mail sender/receiver. An address is associated with sender’s/receiver’s name, room number, and location of the door of the room.

User story A brief description of a piece of system functionality as viewed by a user. It represents something a user would be likely to do in a single sitting at the computer terminal. More information about user story can be found in [51].

Web interface The actual part of the browser where users can view and execute operations on the internet. These operations include: book a delivery, check status of package/mail, personal information management, and system maintenance.

Web server A web server [52] refers to the software and hardware that deliver web pages on the request to clients using the Hypertext Transfer Protocol (HTTP). The sender/receiver/administrator can visit the webpage and book a delivery, request for current status and system management.

3. SYSTEM REQUIREMENTS

3.1. Enumerated functional user stories

We will begin extracting the main points of the Customer Statement of Requirements, and refining all these concepts to obtain the system requirements, that we will present using the more convenient user story syntax. The final list of the most important functional user stories is shown in table 3.1.

Basically, we can see that the main requirement is the package delivery, so let's analyze it in detail. The sender needs to be able to tell the system he/she wants to make a delivery (ST-2). Then, the robot will come to his/her office to pick up the package (ST-5), but before this, the sender needs to authenticate in front of the robot (ST-12¹). Then, the system should notify the receiver that a package/mail is about to be delivered (ST-6) and, after this, the package must be delivered (ST-7), prior receiver authentication (ST-13²).

Of course, we don't want the user to re-enter all its information every time he/she uses the system; that is why we included the register/login subsystem (ST-1). Another convenience for the user is that is easier to remember the name of the receiver than its address, that is why we included ST-3. As specified in the CSR, the sender will be able to use a barcode scanner to input the information to the system (ST-4). Finally, the sender needs to be informed about every step in the delivery (ST-8).

On the other hand, our system will need some maintenance (although not much, as we will discuss later). ST-10 may seem contradictory, because the role of an administrator is to worry about the maintenance of the system. However, this user story should be viewed as: the role of the administrator won't be about looking constantly at the critical parts of the system and check that all works properly. In contrast, the system will notify the administrator if something is going wrong. Also, the administrator needs to have access to all the history of the system (ST-9) and does not want to worry about the charging of the robot (ST-11). It is not clear whether the user story ST-11 is functional or not, because an opposite user story (the administrator will take care about the charging of the mobile parts) would be functional.

Note that we proposed only the user stories containing features that we think we will be able to implement (and test) at the end of the semester. Also, we realize that this list is ambitious and, therefore, we will prioritize the core features and some of the less priority may not be finished.

Again, the most important part of the system is the delivery itself. Therefore, ST-2, ST-5 and ST-7 (the core functionality of our system) are the most priority: we need to have a working system as soon as possible. Then, the convenience of the user is one of the priorities of our customer (because we want the user to use the system, and we need to keep it simple and not to confuse the user with lots of options and tedious processes) and, therefore, ST-3 is also important, followed by the receiver notification (ST-6) and the tracking of current/historical deliveries (ST-8, ST-9). The administrator work must be minimized as soon as possible, for reducing costs (ST-10). Finally, the register/login is not a priority to our customer³. Also, the barcode scanning and the charging of the system will be done as the last step (ST-11).

Note that we didn't assign sizes because some user stories maps to the same system requirement.

¹Non-functional requirement

²Non-functional requirement

³Project description webpage shows this.

ID	PW	User story
ST-1	1	As a sender or a receiver, I can create, manage, modify and login to my user account to be able to use the system.
ST-2	5	As a sender, I can book a delivery without moving from my position to begin the process of transferring a package between different offices.
ST-3	4	As a sender, I can send a package to the receiver based only on his or her name, so that I don't need to care about his or her location.
ST-4	1	As a sender, I can provide the receiver's information to the system by scanning the barcode on the mail or package, so that packages that already contains this information can be processed much easier.
ST-5	5	As a sender, my package will be picked up in my office, so that I don't need to deliver it in a required place or directly to the receiver.
ST-6	3	As a receiver, I want to be notified several minutes prior to the arrival of my package, so that I will be informed about the delivery status.
ST-7	5	As a receiver, my package will be delivered in my office, so that I don't need to go look in another required place (such as my mailbox).
ST-8	3	As a sender, I am able to track the status of my delivery and be notified if problems occur, so that I don't need to worry about.
ST-9	3	As an administrator, I can have access to the record of every delivery and acknowledge, in front of a third person, that the delivery was finished correctly or terminated incorrectly.
ST-10	2	As the administrator, I need to be able to be notified when there are some non-automatically solvable problems. I don't want to care about other problems.
ST-11	1	As an administrator, I don't need to care about the charging of the mobile parts of the system.

Table 3.1.: Functional User stories

3.2. Enumerated non-functional user stories

In this section, we treat non-functional requirements as user stories (see table 3.2), as done in [46]. We regard non-functional requirements as “constraints” we put on the system and each constraint we put on a system narrows the design choices a bit. Trying to put constraints (non-functional requirements) into the user stories format is a good exercise as it provides a better understanding of who wants what and why. Besides, we can have a clear picture about the attributes or characteristics of the system.

We already talked about ST-12 and ST-13 when talking about functional user stories. Although ST-12 and ST-13 are an essential part of our system, we included it in the non-functional user story list because both are very tied to authentication. Also related to security, the user needs to have confidence about the quality of the container, in terms of guarantee that only when the system wants to open it, its contents are available (ST-15).

The user needs to have a little help page available (ST-14), just in case he/she have some issue about the system workflow.

Note that ST-16 is accomplished immediately with the functional requirements proposed, and therefore its weight does not matter: it is a usability requirement.

The rest of user stories are about reliability, cost and performance: the server needs to be power-cut tolerant (ST-17) and condition-independent (ST-19), energy-efficiency of the robot (ST-18), multiple delivery to satisfy user requirements about timing (ST-21) and, finally, the system needs to be affordable (ST-20).

Finally, as we discussed in section 1.3, our system is scalable and extendible (ST-22), which are supportability factors.

ST-21 suggests that there should be a minimum Quality of Service guarantees. Therefore, we need to specify a bussiness policy:

ADS-BP00 - This business policy states that a delivery must be done under certain Quality of Service guarantees. Tentatively, we will set a maximum delivery time of 15 minutes, since the booking of the delivery until the delivery itself.

3.3. Acceptance tests for user stories

Acceptance tests are tests conducted to determine if the requirements of the Auto Delivery System(ADS) is met. Note that acceptance tests are only coarse description about how a user story in ADS is tested. Use Case Acceptance Tests in Section 3.8 will provide fully dressed test description of each use case.

Acceptance Test Cases for ST-1:

- ACT1.01 Ensure any user to be able to open an account with one specific username and password through web or mobile interface (pass);
- ACT1.02 Ensure any registered user to log in to his or her account by inputting correct username and password (pass);
- ACT1.03 Some users can not log in to his or her account by inputting wrong username and password (fail);
- ACT1.04 Ensure any registered user to be able to modify his or her personal account information (e.g. name, email address, office location etc.) through web or mobile interface (pass);

Note that in ACT1.02 and ACT1.04, the word “registered” means the user has successfully executed the step of open an account (as indicated in ACT1.01). We will not test the account management function of administrator’s accounts in this set of test cases.

Acceptance Test Cases for ST-2:

- ACT2.01 An registered user books a delivery by logging in to his or her account and input booking delivery information (pass);

ID	PW	User story
ST-12	3	As a sender, I need to be sure that the receiver will be authenticated before having access to the package, so that I can be sure that only the receiver have access to the package.
ST-13	3	As a receiver, I need to be sure that the sender had been authenticated before sending the package, so that I can know that the sender is who is claiming to be.
ST-14	1	As a sender or receiver, I want to have access to a help page, so that I can understand easily how the system works.
ST-15	3	As a sender, I must be sure that the package sent is, at all moments, in a safe container, and anyone but the receiver will be able to access it.
ST-16	-	As a sender or receiver, I need to be sure that the system maintenance time is less than 1 hour each day.
ST-17	1	As an administrator, I need to be sure about delivery robot is able to keep its booked records safely and correctly stored at 99% of the day even system happens to breakdown.
ST-18	1	As an administrator, I should be able to assume delivery robot can use its energy efficiently and its battery is capable of working as long as 5 hours before recharging.
ST-19	1	As a customer, I want the automatic delivery system to deliver my packets even in the case of different floor or building.
ST-20	1	As a customer, I need to spend less than 500 dollars for purchase and 20 dollars maintenance cost per month.
ST-21	3	As an administrator, I want the system to allow three deliveries at the same and each delivery costs less than ten minutes, so that the users will be happy with time cost and less deployed robots.
ST-22	1	As a customer, I need to be sure that I will be able to extend and scale the solution to a much more useful scenario, such as multi-floor or inter-building deliveries.

Table 3.2.: Non-functional user stories

- ACT2.02 An registered user can not finish booking delivery process online using web interface or mobile interface(fail);

Note that ACT2.01 and ACT2.02 focus on the process of booking delivery online, rather than the previous steps, such as “log in”. By testing whether a booking can be finished online can we assure ST-2 to be implemented.

Acceptance Test Cases for ST-3:

- ACT3.01 In the process of creating a new account, the office location information will be requested from the applicant.(pass);
- ACT3.02 In the process of booking a delivery, the sender arranges the delivery destination location by typing receiver’s name in the booking request.(pass)
- ACT3.03 The robot will deliver the package to the receiver’s correct location(pass)
- ACT3.04 The robot delivers the package to a false location, which is not what the receiver registered in the system.(fail).
- ACT3.05 Any registered user can always update his or her location information to the system.

Note that each user is able to manage their account information, so it is the user’s obligation to update his or her location into the system when his or her office location has changed. Only in this way can the delivery to be more accurate.

Acceptance Test Cases for ST-4:

- ACT4.01 The robot will correctly send package to the receiver after the sender scanned the package and put it into the robot(pass);
- ACT4.02 The system does not identify the package information when scanning the barcode of the package (fail);

Note that ST-4 will not be implemented in this semester’s project because of the necessity of decreasing the complexity of the system. As a result, this test case should only be tested in the future when this corresponding user story is implemented.

Acceptance Test Cases for ST-5:

- ACT5.01 A robot arrives at the sender’s office to pick up the package for delivering in 5 minutes after booking a delivery (pass);
- ACT5.02 The sender (who has already booked a delivery online) does not see the robot coming for picking up packages in 5 min (fail)
- ACT5.03 The sender puts the package in the compartment of the robot and the robot leaves for delivery to the receiver (pass);

Note that in this set of test cases features regarding “authentication to robot before a sender puts in his package” and “robot moves from one place to another” will not be tested, even though these features is of great importance in the process of pickup packages. Detailed testing regarding these two features will be mentioned in section 3.8.

Acceptance Test Cases for ST-6:

- ACT6.01 A receiver will receive a notification email regarding his or her package to be delivered, at the time robot picks up package from the sender and starts to deliver (pass);
- ACT6.02 A receiver does not receive any notification email regarding the package before the robot shipped the package to his or her office (fail)

Note that the email will be sent during the time slot that sender has put the package into a compartment of the robot and the robot is about to leave for delivery, so what ACT6.01 indicates about the prior notification will be how much earlier depends on how long it will take for the robot from the sender’s location to the receiver’s location. Tentatively we make it to be 5 minute.

Acceptance Test Cases for ST-7:

- ACT7.01 The robot arrives at the receiver's office in 5-10 minutes after picking up package from the sender's location (pass);
- ACT7.02 The robot arrives at the receiver's office more than 10 minutes without a miss between the robot and the receiver (fail)

Note that the time described in ACT7.01 and ACT7.02 is actually determined by the distance between sender's office and the receiver's office, as well as the average speed of the robot. Also, the term "miss" in the ACT7.02 indicates the scenario that the robot arrives at the receiver's office but the receiver is not available at the moment.

Acceptance Test Cases for ST-8:

- ACT8.01 A sender checks the accurate status of his or her previous delivery request, the normal status includes "Waiting for Pickup" "Delivery To Receiver" etc. and its corresponding execution time (pass)
- ACT8.02 A sender receives email notification when the package delivery fails and is about to return to the sender (pass)
- ACT8.03 A sender receives inaccurate status report in the account system.(fail)

Note that this set of test cases focus on how a sender can track the status of the delivery, rather than previous steps of online operation, such as login, choose "delivery status check" operation. The inaccurate status includes when there is returns in the delivery but the sender is not notified about this issue, or another instance that when a package is delivered but the status still show "delivering" status, and so forth.

Acceptance Test Cases for ST-9:

- ACT9.01 An administrator logs into the system and checks all delivery information in the system, sorted by current delivery and history delivery information(pass)
- ACT9.02 An administrator can only check part of the users' delivery information (fail)

Note that the administrator has higher priority that can see all users' delivery history and current delivery information. The delivery information that the administrator can see is the same as what a user can see, including delivery process, pickup time, delivery time, attempt of delivery and so forth.

Acceptance Test Cases for ST-10:

- ACT10.01 An administrator receives an email when the robot is out of track (pass)
- ACT10.02 An administrator receives an email when the robot is out of power and cannot connect with charger immediately (pass)
- ACT10.03 An administrator receives an email when a package is returned to the sender but after three times of return delivery attempt the receiver is still not available;(pass)
- ACT10.04 An administrator receives an email when the robot is out of control by the central station(pass);
- ACT10.05 An administrator receives an email when system crashes and need to be maintained(pass)
- ACT10.06 Any above exceptions happens when the administrator does not receive any notification email about that (fail)

Note that the email that the administrator receives includes the category of the exception and the advice handling with that issue, as well as with the time that the unexpectation happens.

Acceptance Test Cases for ST-11:

- ACT11.01 Ensure the robot to be recharged in recharging station every time the battery is lower than 20
- ACT11.02 The robot runs until out of battery (fail)

Note that the security of charging need also be considered, but it is not mentioned in this test case, because ST-11 is a functional user-story, and security reason, such as the temperature, voltage of the battery etc.

Acceptance Test Cases for ST-12:

- ACT12.01 The receiver inputs his or her account password as an authentication to open the compartment for getting packages (pass)
- ACT12.02 The receiver opens the compartment and take out the package without entering the password (fail).

Note that in the actual implementation, we use a LED with green-red lights as a lock to each compartment of the container of the robot, in order to decrease the complexity. When a LED is green, it shows that the cover of the compartment is open and the receiver is able to take out the package. When the LED turns red, it shows that the compartment has been locked and no one could take out anything from the compartment in the robot.

Acceptance Test Cases for ST-13:

- ACT13.01 The sender inputs his or her account password as an authentication to open the compartment for putting in packages (pass)
- ACT13.02 The sender puts in the package in one of the compartment in the robot without inputting any password (fail)

Note that when a LED is green, it shows that the cover of the compartment is open and the sender is able to put in the package. When the LED turns red, it shows that the compartment has been locked and no one could put in anything into the compartment in the robot.

Acceptance Test Cases for ST-14:

- ACT14.01 Any user is able to open the help page on the web interface or mobile interface(pass)
- ACT14.02 Any user is able to search key words in the search bar of the help page.
- ACT14.02 Help page is not accessible by web interface or mobile interface(fail)

Note that the help page includes basic instructions about how to use the system. A search bar is available for user to search information.

Acceptance Test Cases for ST-15:

- ACT15.01 Any compartment of the container in the robot is kept close all the time unless after authentication to the robot by a sender or receiver (pass)
- ACT15.02 When the LED turn green at the time where no sender or receiver ever authenticates, the system will notify police and administrator.(pass)

Note that ACT15.01 and 15.02 focus on safe guard of each compartment – LEDs. A compartment is regarded as open when LED on it turns green, and is regarded as closed when LED turns red.

Acceptance Test Cases for ST-16:

- ACT16.01 Maintenance time of each day is less than one hour (pass)
- ACT16.02 Maintenance time of each day is more than one hour(fail)

Note that we assume that if maintenance time per day is lower than one hour, it means user do not need to worry about the time that the system is not in service. The serving time of each day will generally be determined by weather, office condition, battery and power supply, as well as maintenance time. So the time of maintenance every day might be subject to change.

Acceptance Test Cases for ST-17:

- ACT17.01 All user information and data can be restored after system breaks down due to backup of user data (pass)
- ACT17.02 User Information lost in severe breakdowns of system (fail)

Note that the system will use backup mechanism to prevent system unexpectedly shutting down or crashing.

Acceptance Test Cases for ST-18:

- ACT18.01 In the case of full load, the robot can work as long as 5 hours before charging battery(pass)
- ACT18.02 In the case of full load, the robot can work less than 5 hours before charging battery (fail)

- ACT18.03 In the case of idle status, the robot can stay as long as 10 hours before charging battery(pass)
- ACT18.03 In the case of idle status, the robot can stay less than 10 hours (fail)

Note that in ACT18.01 and ACT18.03, “full load” means the robot is moving all the time, as well as signals communicating with central station; In ACT18.02 and ACT18.04, “idle status” means the robot rarely moves, and the time slot of signal communication becomes very long.

Acceptance Test Cases for ST-19:

- ACT19.01 Ensure delivery process to be completed when sender and receiver are in different floors of the same building (pass)
- ACT19.02 Ensure delivery process to be completed when sender and receiver are in different buildings (pass)
- ACT19.03 Ensure delivery process to be completed when bad weather happens(pass)

ST-19 might not be implemented in this project, considering what we can achieve in a full semester. In the case ST-19 can be developed, ACT19.01 and ACT19.02 should be tested.

Acceptance Test Cases for ST-20:

- ACT20.01 Ensure delivery process to be completed when sender and receiver are in different floors of the same building (pass)
- ACT20.02 Ensure delivery process to be completed when sender and receiver are in different buildings (pass)
- ACT20.03 Ensure delivery process to be completed when bad weather happens (pass)

Note that ST-20 might not be implemented in this project, considering what we can achieve in a full semester. In the case ST-19 can be developed, ACT19.01 and ACT19.02 should be tested.

Acceptance Test Cases for ST-21:

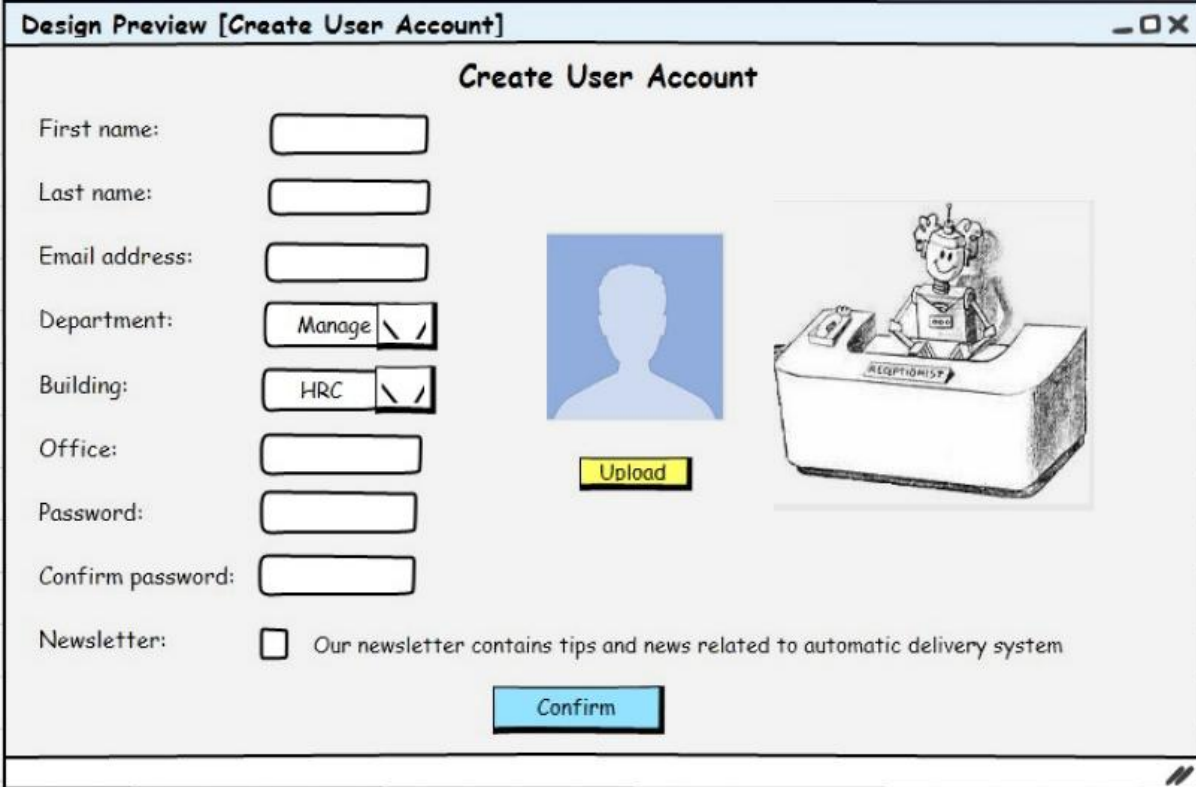
- ACT21.01 One-time cost of the system will be less than 1000 dollars(pass)
- ACT21.02 Daily cost of the system will be less than 5 dollars(pass)

Note that the one-time cost includes cost of the central system, robot, setting the website and mobile interface, and laying out tracks. Daily costs contains electric costs and maintenance cost.

Acceptance Test Cases for ST-22:

- ACT22.01 Schedule multiple delivery requests as first come, first server basis, more requests should wait in line (pass);
- ACT22.02 When delivery and Pickup happened at the same time, delivery comes prior to Pickup (pass)
- ACT22.03 A Sender can give multiple packages to robot at the same time (pass)
- ACT22.04 A receiver can get multiple packages at the time robot arrives at his or her office (pass)
- ACT22.05 A receiver receives packages that do not belong to him or her(fail)

Note that in the case of multiple delivery at the same time, ACT22.01 tests how the system will handle with multiple book delivery request; ACT22.02 tests with how the system will react when receiver of last delivery case and sender of current case happen to be the same person; ACT22.03 tests with how the system will handle when the sender has multiple packages waiting to be pick up; ACT22.04 tests with the case that an receiver receives multiple packages at the same time.



The screenshot shows a web browser window titled "Design Preview [Create User Account]". The main heading is "Create User Account". The form contains the following fields and elements:

- First name:
- Last name:
- Email address:
- Department: (dropdown arrow)
- Building: (dropdown arrow)
- Office:
- Password:
- Confirm password:
- Newsletter: Our newsletter contains tips and news related to automatic delivery system

There is a blue silhouette placeholder for a profile picture with a yellow "Upload" button below it. To the right is a cartoon illustration of a robot receptionist at a desk labeled "RECEPTIONIST". A blue "Confirm" button is at the bottom center.

Figure 3.1.: Create user account

3.4. On-screen Appearance Requirements

Figures 3.1, 3.2, 3.3, 3.4, 3.6 and 3.5 show the screen appearance scheme that the customer provided to us. These figures describes the appearance of the user account creation, the delivery history, the appearance of checking the status of a concrete delivery and how to modify an account information.



Figure 3.2.: Delivery book

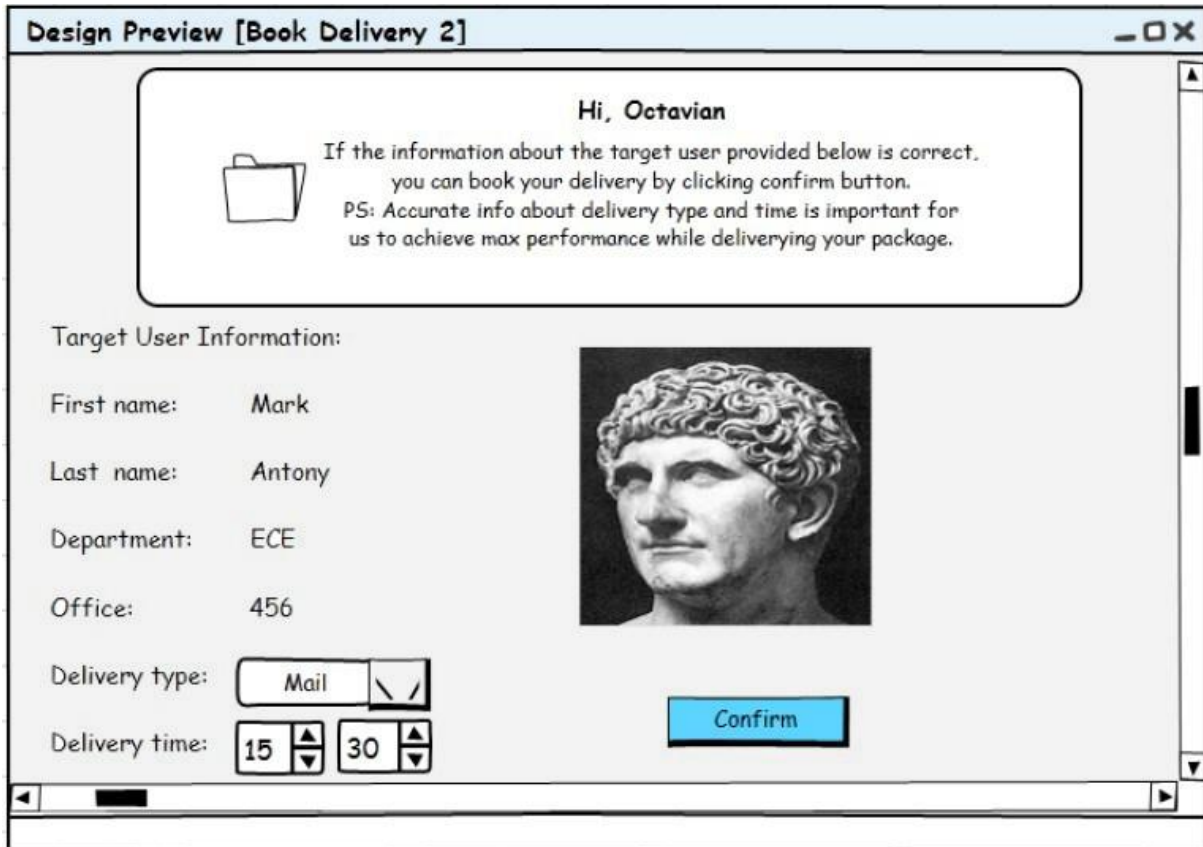


Figure 3.3.: Delivery book

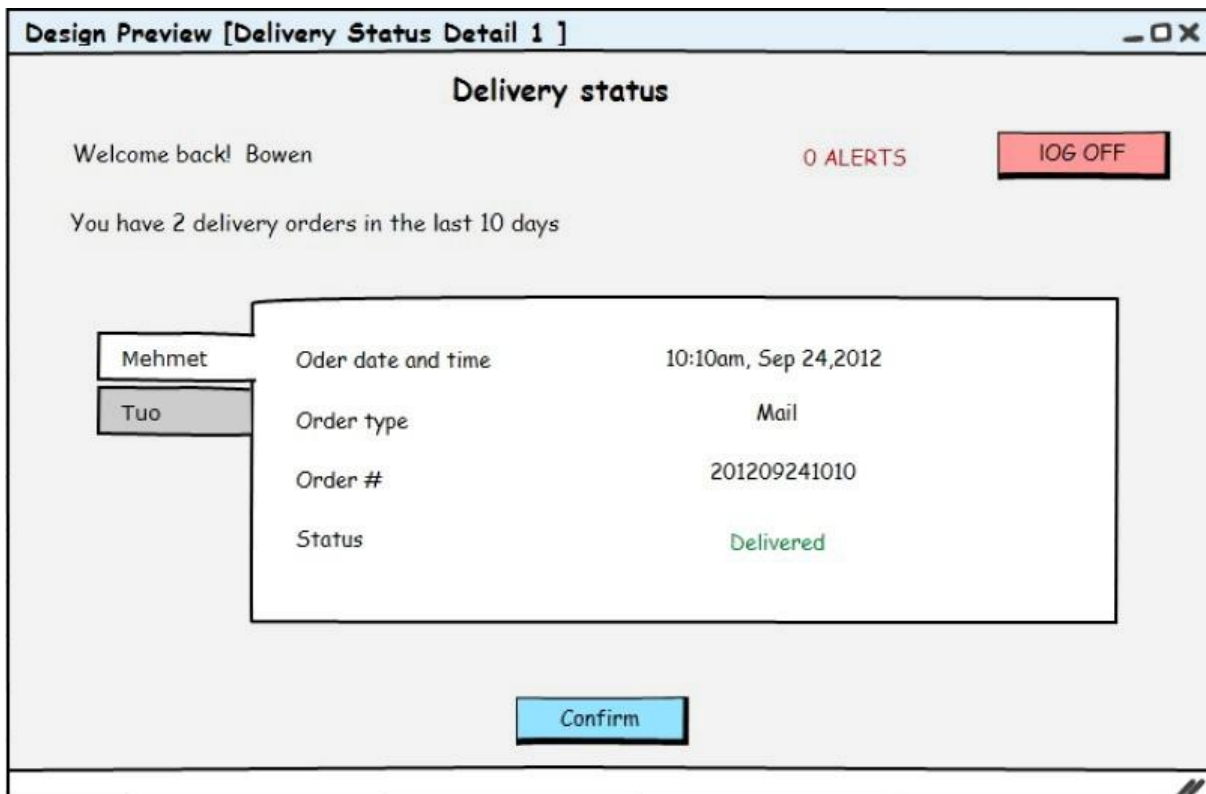


Figure 3.4.: Delivery status

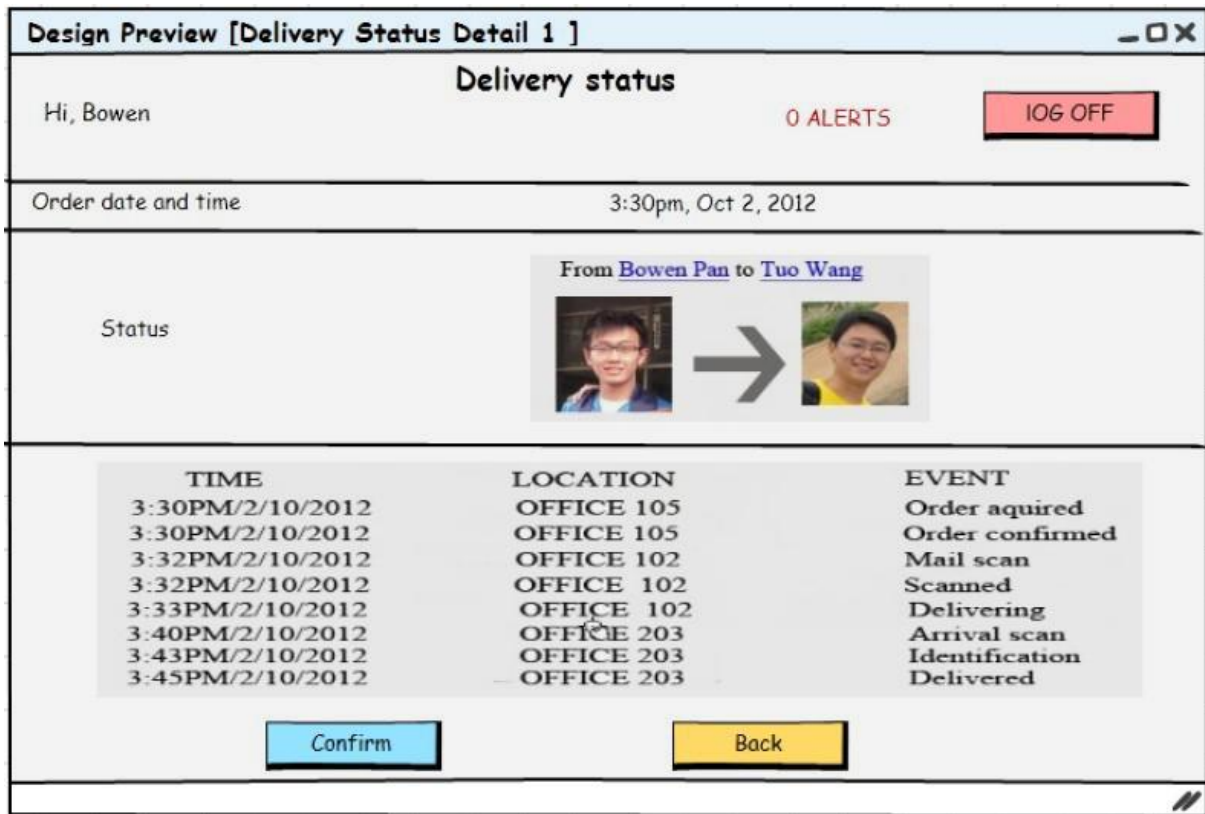


Figure 3.5.: Delivery status



Figure 3.6.: Modify user info

4. FUNCTIONAL REQUIREMENTS SPECIFICATION

4.1. Architectural style

Different global architectures of the system gives different user experience to the system. Therefore, we will begin describing the architectural style of the system. The system will be composed of 3 main components: the central station, the robot and the location marks.

1. The **central station** will be the responsible, basically, of:
 - a) keeping track of the robot location
 - b) storing the user's information (such as room, name, username, password)
 - c) serving the web page to the users and the android application back-end and receive user's feedback
 - d) keeping track of each individual delivery process (origin, destination, time to pick up, time to deliver...)
 - e) planning pick up processes and deliveries to optimize the path
 - f) sending e-mails to the senders/recipients when the robot is approaching their room
 - g) sending e-mails to the maintenance team when a hazardous situation occurs
2. The **mobile robot** will be the responsible, basically, of:
 - a) waiting for the central station to request a movement to a certain point
 - b) moving from one point to another one following the location marks
 - c) to provide a numerical keyboard to allow user authentication
 - d) to provide a secure sectioned box with a secure locked lid in every section
 - e) to provide a barcode scanner to the user
3. The development team will place convenient **location marks** to:
 - a) indicate the robot which path to follow. For this purpose, we will use black and white adhesive tapes.
 - b) indicate the robot which are the most notable points (such as rooms or the charging station). For this purpose, we will place some magnets strategically in order to allow the robot locate such important points.

In order to implement the central station we will need a web server and a database management system. We plan to use web-services to provide a convenient interface to the back-end application and a regular dynamic webpage/android application that will call them. We will need regular mechanisms (such as mail or mudd unix commands) to send e-mails automatically.

In order to implement the robot, we will use a structure similar to a simple remote control car with a little sectioned box on top of it. For our project, we won't buy a professional box with lids that can be automatically locked and unlocked. The process of opening and closing each concrete lid will be simulated by a led in front of each section of the box: a green light will mean that the lock have been released, and no-light will mean that the lid is locked (and, therefore, we can assume that the content is secure). Our robot will be automated using an Arduino Uno [5], an Arduino Wifi shield [6] and possibly an Arduino Motor controller shield [4].

4.2. Stakeholders

The stakeholders of our system are:

End user The end user can be both the sender or the receiver of a package, and have a direct interest in our system: use it, its functionality.

Customer The customer can be the organization who will deploy our system, and is interested in its cost and the timeline to deploy it.

Administrator The administrator can be the individual or team who will take care that the system is working properly, and will be in charge to solve the issues with the system (or contact the supplier, if the issue is not easily solvable).

Supplier The supplier (our team) will be the manager, the architect, the developer and the installer of the system.

4.3. Actors and goals

The actors of our system can be splitted between human and non-human. The human actors are the sender, the receiver and the administrator. The non-human actors are the two motors and the four sensors:

User The end user of the system.

Sender The end user that will use the system to send a package/mail.

Receiver The end user that will receive a package/mail with our system.

Administrator The manager that will be in charge of keeping the system in working order.

Timer The timer will control the schedule of the robot and make it go to pick up at a certain point or delivery in another one.

MotorL The left motor of the robot

MotorR The right motor of the robot

ReflectiveOpticalSensorL Along with ReflectiveOpticalSensorR, will detect the path that the robot needs to follow.

ReflectiveOpticalSensorR Along with ReflectiveOpticalSensorL, will detect the path that the robot needs to follow.

MagneticSensorL Along with MagneticSensorR, will detect the position of the robot.

MagneticSensorR Along with MagneticSensorL, will detect the position of the robot.

BatteryLevelIndicator Will detect the charging level of the robot's battery.

StatusLEDs Will emulate the individual lock mechanisms in the segmented secure correspondence container.

Buzzer Will be equipped in the robot and will be useful to emit sounds.

The User actor may seem a broad term to the reader. However, we think that it must be included as a parent of Sender and Receiver because these two specific actors can share some features.

On the other hand, we included the Timer because, as we will discuss later, it will initiate full use cases.

4.4. Use case diagram

In figure 4.1 we show the use case diagram for our system.

An approach to gather the use cases from the user stories is to find a single virtual use case that describes the entire flow of the usage (sketchy use cases). The coarse-grain use case for our system is the



Figure 4.1.: Use case diagram.

action of delivering a package from a sender to the receiver. Of course, we need to decompose this broad virtual use case in several real use cases.

One of the most important decisions that we did when designing the system was to separate this big use case in, basically, 3 smaller ones: BookDelivery, PickUpPackage and Delivery. Although all three must be performed in order, we decided that not necessarily consecutively. In other words, if three senders book a delivery, the robot can pick up two of them, delivery one of them, pick up another one and finally delivery two, if this sequence reduces the path.

Note that although the NotifyError may seem to be initiated by a timeout interval, we think that Timer is not the correct actor to initiate it, because the timeout interval will be controlled by the system (use case MoveRobotToPoint).

Note that the ManageUser use case can be used by three actors: the sender, the receiver (both are the end user) and the administrator. However, the system will behave different depending on who is initiating the use case. If the sender or the receiver are initiating it, the system will allow to manage their own account. If the administrator is invoking it, the system will allow to manage all the accounts. Note also that this use case have three extensions to add, edit or remove a user.

Although we know that a use case describes the interaction between an actor and the system (and, therefore, an actor would need to initiate it), we included several second tier use cases that are not initiated by any external actor, but are included in (and, therefore, initiated by) other use cases. We specified these use cases to refine or detail a bit more the interaction between use cases.

First, we thought that the use cases SetMotorSpeedX and PositionInspection were too fine-grained to be included in our diagram, and should be individual steps in the MoveRobotToPoint use case. However, we divided MoveRobotToPoint in three use cases because (i) we think that this use case is very huge in comparison with the others and, (ii) we thought that having many actors related to a single use case would be confusing.

Note that we decided to use inheritance (in front of extension) as the relationship between ManageUser

and AddUser, EditUserInformation and RemoveUser because the 3 latter share common functionality.

4.5. Use cases casual description

Following we describe the actual use cases that we detected:

UC-1 BookDelivery Allows the sender to book a delivery. This is the action that begins the sketchy coarse-grain use case that we mentioned.

UC-2 TrackDeliveryStatus Allows sender and receiver to check the delivery status of package or mails. By logging in the system, sender and receiver can check their delivery histories and can have an access to any delivery to see detailed delivery information, such as the package location and corresponding status at each specific time.

UC-3 ObtainHelp Allow sender or receiver to obtain help using web interface or mobile interface. Derived from user story ST-14.

UC-4 ManageUser It allows the administrator to get access to modify the system account settings like to add user, remove user and to edit user information. The administrator can log in only if he or she provide the correct and administrative-level account. Extension point: the administrator has a option to further add user, remove user or edit user information. Derived from user stories ST-1

UC-5 AccessDeliveryRecord To check the delivery status and history (for user and administrator) and edit the record if necessary (administrator only).

UC-6 Login To log into the system as an authenticated person.

UC-7 PickUpPackage Allows a sender to send his or her package right at his or her office. Derived from User Stories: ST-5, ST-9, ST-15;

UC-8 Delivery Allows the system to notify receiver and make mobile part of the system (robot) move from sender's location to the corresponding receiver's location, authenticate receiver and give the package to the receiver.

UC-9 EditUserInformation It allows the administrator to modify the user information after he or she logs in the system with valid account. The administrator can update the newest information of Email and office address and so on. It's crucial for nowadays position and other information change is frequent in the office. (optional sub use case, extend UC-: ManageUser) Derived from user stories ST-1

UC-10 AddUser It allows the administrator to create a new user account after he log in with valid account. The administrator should input the information like First Name, Last name, Email address and Office address to finish add a new user account (optional sub use case, extend UC-: ManageUser) Derived from user stories ST-1

UC-11 AuthenticationToRobot Allows a sender/receiver to be authenticated before sending the package or mail. A sender/receiver needs to provide credentials of a valid password before he or she deliver/receiver the package or mail by/from the robot. For simplicity, the password will be the same as the sender/receiver's login password. An invalid entry will result in an error notification and failure of opening the mailbox, and the sender/receiver will be prompted again for password entering. When the count of failed attempts exceeds the maximum allowed number, an error notification will be made and this delivery/receiving task failed.

UC-12 NotifyReceiver Allows a receiver to be notified about the delivery several minutes before the arriving of a package or mail. The notifying message is sent to the receiver automatically. This is available only to registered receivers.

UC-13 NotifyError To notify the administrator and users there are errors occur in the delivery system.

UC-14 ChargeBattery Allows the system to automatically charge the battery of the delivery machine. For the system to be robust and autonomous, automatic battery charging is critical. A battery level indicator is one actor (initiator) such that its indication is used as an input to the system to decide when to activate and use this functionality. Another actor associated with this functionality

is StatusLEDs which will be used to show the system is in the automatic battery charging state. Derived from user story ST-13.

UC-15 MoveRobotToPoint The system will need to move the robot from its current position to a given point. This use case allows the system to move the robot from one point to another one, if other use cases requires it.

UC-16, UC-17; SetMotorLSpeed, SetMotorRSpeed In order to move a robot, the system will need to specify which is the speed of each motor in the robot (left-right). This use case describes the interactions to achieve this goal.

UC-18 PositionInspection While moving the robot from one point to another one, the system will need to check whether it is in the final position or not. This use case will accomplish this function.

UC-19 RemoveUser It allows the administrator to delete the user account after he or she logs in the system with valid account. It's necessary for administrator to update the newest information if some people leaves the position. (optional sub use case, extend UC-4: ManageUser) Derived from user stories ST-1

4.6. Traceability matrix

Req	PW	UC7	UC12	UC8	UC2	UC3	UC1	UC5	UC4	UC10	UC9	UC19	UC15	UC11	UC6	UC13	UC16	UC17	UC18	UC14
ST-1	1								x	x	x	x			x					
ST-2	5						x													
ST-3	4						x													
ST-5	5	x											x				x	x	x	
ST-6	3		x																	
ST-7	5			x									x				x	x	x	
ST-8	3				x															
ST-9	3							x												
ST-10	2															x				
ST-11	1																			x
ST-12	3													x						
ST-13	3													x						
ST-14	1					x														
Max PW		5	3	5	3	1	5	3	1	1	1	1	5	3	1	2	5	5	5	1
Total PW		5	3	5	3	1	9	3	1	1	1	1	10	6	1	2	10	10	10	1

Table 4.1.: Traceability matrix

The traceability matrix is shown in table 4.1. Note that we included some non-functional requirements (ST-12, ST-13 and ST-14) because they all have some use case associated.

The use cases can be sorted as follows, in terms of priority:

$$\text{UC15, UC16, UC17, UC18} > \text{UC1} > \text{UC11} > \text{UC7, UC8} > \text{UC12, UC2, UC5} > \text{UC13} > \text{UC3, UC4, UC10, UC9, UC19, UC6, UC14}$$

As we can see, the most important use cases are the ones that makes the robot move and go from one point to another one. This is easily explainable, as these part is related for both picking up and delivering a package. Following these, the BookDelivery use case and, farther, the AuthenticationToRobot use case.

4.7. Use cases fully-dressed description

4.7.1. BookDelivery

Table 4.2 shows the details of the BookDelivery use case, and the flow of events or interactions is defined as follows:

1. include::Login(UC-6)
2. ← **System** shows a table of options, including “book a delivery”.

BookDelivery	
Related requirements	ST2, ST3
Initiating actor	Sender
Actor's goal	To schedule a delivery process and specify the receiver/receivers.
Participating actors	-
Preconditions	The sender is a registered user
Success end condition	One or several deliveries have been scheduled, and registered in the schedule database.
Failed end condition	User cancelled the booking or failed to enter the required data before timeout interval.

Table 4.2.: Summary for the use case UC-1: BookDelivery

3. → **Sender** selects the option “book a delivery”.
4. ← **System** shows (i) a screen with a list of the *starred* receivers by the sender, (ii) a searchbox for searching the receiver by name or by office/building.
5. **Sender**
 - a) → selects a receiver from the *starred* list
 - b) → searches a receiver using a name, or a portion of it
 - i. **System**
 - A. 1. ← finds and returns more than one result to the search
 2. → **User** selects one receiver.
 - B. 1. ← does not find any match and returns an error telling this.
 2. → **User** clicks continue
 3. ← **System** go to step 4.
 - c) → searches a receiver using its office and building
 - i. **System**
 - A. 1. ← finds and returns more than one result to the search
 2. → **User** selects one receiver.
 - B. 1. ← does not find any match and returns an error telling this.
 2. → **User** clicks continue.
 3. ← **System** go to step 4.
6. ← **System** shows a list of estimated pick up times (in HH:MM format)
7. → **User** selects its desired pick up time.
8. ← **System** asks if the user wants to book more deliveries.
9. **User**
 - a) → wants to book another delivery.
 - i. ← **System** adds the current book delivery in a temporary buffer and returns to point 4.
 - b) → does not want to book another delivery.
 - i. ← **System** shows a confirmation screen with the list of all the booking delivery requests in the temporary buffer.
 - ii. **User**
 - A. 1. → confirms
 2. ← **System** books all the temporary requests and goes to the main menu (step 2).
 - B. 1. → discards the bookings
 2. ← **System** discards all the requests and goes to the main menu (step 2).

Comments

Note that the main success scenario (book a single regular delivery) is composed by:

1 2 3 4 5a 6 7 8 9b.i 9b.ii.1 9b.ii.2

This corresponds to 6 different screens to book a delivery¹. However, as we described the interaction between the System and the Sender, we do not necessarily need to implement it this way. Moreover, we will reduce the user experience to the minimum: after the login, the booking screen will appear (booking is the main usage of the web/mobile interface) with a list of starred receivers and the 2 search boxes. The user will click on the receiver, probably in the starred list, and the name will move to the first entry of an empty table (booking table). The time to be picked up will be ASAP (as soon as possible) by default. To confirm the single delivery, the user only needs to click to the confirm button at the end of the page. We reduced the experience to 3 clicks² and 2 screens. To book multiple deliveries, instead of pressing the global confirm, the user will press the confirm button at the end of the current row (1st row, in the example), in the booking table. Then, the user will be allowed to choose another receiver (without changing the screen) and repeating the process until the global confirm is pressed.

Notes

In the step 4 the concept of *starred* receivers of a given sender is introduced. However, a Business Policy must be defined to precise its meaning:

ADS-BP01: this Business Policy specifies that the *starred* contacts will be composed by the union of the set of the 5 most recently contacted and the set of the 5 that have been contacted more times.

The whole step 5c) does not specify where (at which database) the system looks at the office/building introduced by the sender. This is clarified by the next Business Policy:

ADS-BP02: this Business Policy specifies that, given an existing address (building and office), if there is no receiver associated to it, the address can't be used to book a delivery.

The loop in step 9.a.i) can be infinite. This can be exploited by a malicious user to send repeatedly book deliveries and make the buffers of the server grow and grow. As this can cause issues such as DoS [27], we need to specify a Business Policy to prevent this:

ADS-BP03: this Business Policy specifies that, one sender can only have 50 non-served book deliveries.

ADS-BP04: this Business Policy specifies that the sender have 30 minutes to book all deliveries. If this timeout has gone by, the session will expire.

ADS-BP05: when a delivery is booked, a timeframe is specified. The system will begin moving the robot for picking up the package after $timeframe - T_{AB}$, where T_{AB} is a constant specified during the deployment of the system specifying the time between points A (in this case, current point of the robot) and B (in this case, sender's office).

4.7.2. MoveRobotToPoint

Table 4.3 shows the details of the MoveRobotToPoint use case, and the flow of events or interactions is defined as follows. Is important to note that this case is designed to be re-entrant (i.e. to be initiated as many times as wanted in parallel).

1. → The use case is initiated passing a reference to the destination point (p_d), which can be either: the user parameter (the sender address or the receiver address) or the initial point (if invoked just when the last delivery is finished).
2. **System** checks current robot position p_c in the database.
 - a) If the robot is in the initial position, system continues to step 3.
 - b) If the robot is not in the initial position (i.e. it is moving)

¹Login - Book - Starred - Time - No multiple - Confirm

²Login - Starred receiver - Confirm

MoveRobotToPoint	
Related requirements	ST5, ST7
Initiating actor	Other use cases will include this.
Actor's goal	To specify the destination address.
Participating actors	-
Preconditions	The destination address exists in the database. The robot must be correctly aligned to the track before the first initialization of this use case.
Success end condition	The robot reaches the destination.
Failed end condition	The robot can't reach the destination in a timeout interval.

Table 4.3.: Summary for the use case UC-15: MoveRobotToPoint

- i. If the robot is already going to a position p_{d1} and the new destination (p_d) is between the current position (p_c) and p_{d1} , the system delays the movement to p_{d1} and continues to step 3.
 - ii. Else, the system waits until the robot finished the current positioning and then continues to step 2.
3. → **System** checks if the robot is in the destination point (using use case PositionInspector).
 - a) ← If yes, both motors' speed is set to zero by **System** (using use cases SetMotorLSpeed and SetMotorRSpeed) and continue to step 7.
 - b) If not, continue on step 4.
 4. → **System** checks if the sensors are aligned to the track (using use case PositionInspector).
 - a) ← If yes, both motors' speed is set to its maximum value (using use cases SetMotorLSpeed and SetMotorRSpeed).
 - b) ← If not and is disaligned to the right, decrease the speed of left motor and set right motor speed to maximum (using use cases SetMotorLSpeed and SetMotorRSpeed).
 - c) ← If not and is disaligned to the left, decrease the speed of right motor and set left motor speed to maximum (using use cases SetMotorLSpeed and SetMotorRSpeed).
 - d) ← If not, and the alignment haven't been found for a specified amount of time, try to recover the path.
 5. **System** waits a specified amount of time.
 6. Go to step 3.
 7. The system instructs the **Buzzer** to make a sound to notify the user (**Sender** or **Receiver**).
 8. If some position was delayed in step 2, resume it (go to step 3).

Comments

Note that the main success scenario (book a single regular delivery) is composed by:

1 2a [3b 4a 5] [3b 4a 5] [3b 4a 5] ... [3b 4a 5] 3a 7 8

Notes

Note that this flows specifies that the robot will need to run some path-recovery algorithm. As this algorithm can be only refined testing it in the concrete environment, we will specify this part in a next stage of the product lifetime.

In the step 5 there is a unespecified amount of time:

ADS-BP06: this Business Policy specifies that the amount of time to wait between to accesses at the value must be smaller than 1/10 seconds, but not too smaller, because noise could interfer to the value of the sensors.

PositionInspection	
Related requirements	ST5, ST7
Initiating actor	Other use cases will include this.
Actor's goal	To check the position of the robot.
Participating actors	-
Preconditions	The system can access the sensors in the robot.
Success end condition	The position is checked correctly.
Failed end condition	-

Table 4.4.: Summary for the use case UC-18: PositionInspection

SetMotorSpeedL/R	
Related requirements	ST5, ST7
Initiating actor	Other use cases will include this.
Actor's goal	To change the speed of the left/right motor of the robot.
Participating actors	-
Preconditions	The system can access robot's motor. It is working properly.
Success end condition	The speed of the motor is correctly set.
Failed end condition	-

Table 4.5.: Summary for the use case UC-16/17: SetMotorSpeedL/SetMotorSpeedR

4.7.3. PositionInspection

Table 4.4 shows the details of the PositionInspection use case, and the flow of events or interactions is defined as follows.

1. **ReflectiveOpticalSensorL** and **ReflectiveOpticalSensorR** provides its value to the System.
2. **MagneticSensorL** and **MagneticSensorR** provides its value to the System.

Comments

Note that there is no failure case, because the precondition specifies that the use case should begin only if the system can access the sensors

As this simple approach can't limit the effects of noise, we plan to refine it when defining the system sequence diagrams.

4.7.4. SetMotorSpeedL/SetMotorSpeedR

Table 4.5 shows the details of the SetMotorSpeedL and SetMotorSpeedR use cases, and the flow of events or interactions is defined as follows.

1. **System** specifies the speed that wants to set to the motor.
2. **MotorL** or **MotorR** is set to the specifies speed.

Comments

Note that the speed regulation control may require special treatment depending on the motor device used in the robot. A typical technique in these kind of applications is the Pulse Width Modulation (see [47]).

ChargeBattery	
Related requirements	ST-13 stated in Table 3-1.
Initiating actor	BatteryLevelIndicator.
Actor's goal	To notify the system about the delivery machine battery charge level once this level is below the specified critical limit by sending warning messages.
Preconditions	The delivery machine should spend its batter charge until charge level goes below the limit.
Postconditions	The delivery machine state is affected by the change of its task order.

Table 4.6.: Summary for the use case UC-14: ChargeBattery

Login	
Related requirements	ST1, ST2, and ST11 stated in Table 3-1
Initiating actor	Any of: sender, receiver, administrator.
Actor's goal	To log into the system as an authenticated person.
Participating actors	Keyboard, Panel, Displayer, Operating Interface, names/keys stored in the system database.
Preconditions	The valid names/keys stored in the system database is non-empty; The system operating interface works normally. The user has already created a user account in the system.
Postconditions	Data can be added or altered by user logs into the system.

Table 4.7.: Summary for the use case UC-6: Login

4.7.5. ChargeBattery

Table 4.6 shows the details of the ChargeBattery use case, and the flow of events or interactions is defined as follows. Is important to note that this case is designed to be re-entrant (i.e. to be initiated as many times as wanted in parallel). Flow of Events for Main Success Scenario:

1. → BatteryLevelIndicator detects the battery of the delivery machine to be lower than two different previously decided charge limits and sends a corresponding warning message to the system.
2. ← Once the system takes the message sent by BatteryLevelIndicator, it is going to give a decision about what to do next. This decision stage is important because if the warning is received while the delivery machine is currently doing some given tasks, task of automatic charging of delivery machine battery needs to be scheduled carefully by considering the remaining battery life and the urgency of the current tasks.

We need to be more specific about the “decision stage” of the system for the battery level warning. First battery warning message (BWM 1) will be the indication of battery is going to finish in 15 minutes. This warning is going to be used to give some time to order the delivery tasks and automatic charging of the battery. Second battery warning message (BWM 2) will be the indication of battery is about to be finished. Once the system gets this message it should make the delivery machine to quit its current task and immediately go to the specified charging area.

4.7.6. Login

Table 4.7 shows the details of the Login use case, and the flow of events or interactions is defined as follows.

Flow of Events for Main Success Scenario:

Access Delivery Record	
Related requirements	ST2, and ST11 stated in Table 3-1.
Initiating actor	Any of: sender, receiver, administrator.
Actor's goal	To check the delivery status of current delivery or check the delivery history of the past deliveries.
Participating actors	Keyboard, Panel, Displayer, Operating Interface, Delivery records stored in the system database.
Preconditions	<ul style="list-style-type: none"> • The set of delivery records stored in the system database is non-empty and correct. • The user/administrator successfully log into the system. • The system operating interface works normally.
Postconditions	Delivery records can be displayed chronologically in the displayer. <ul style="list-style-type: none"> • The administrator can edit the delivery records in the system..

Table 4.8.: Summary for the use case UC-5: Access Delivery Record

1. → Sender/Receiver/ Administrator use computers, cell phones or other device connected to the internet. (a) Sender/Receiver inputs correct user name and password and log into the system as a user successfully. (b) Administrator inputs “admin” and special code to log into the system as an administrator.
2. ← System (a) verifies that the key is valid, and (b) signals to the actor the key validity.

Flow of Events for Extensions (Alternate Scenarios):

1. → Sender/Receiver inputs incorrect user name and password.
2. ← System (a) detects error, (b) marks a failed attempt, (c) signals to the actor, (d) if the failed times equal three, block the login dialog box for the actor.
3. → Sender/Receiver (a) supply a valid user name/password, (b) if the login dialog box is blocked, contact the administrator.

The user story of ST1, ST2, and ST9 stated in Table 2.1 require that user or administrator can access to the user account, so we should decide that anyone who want to access the user account should be authenticated by log into the system.

4.7.7. Access Delivery Record

Table 4.8 shows the details of the Access Delivery Record use case, and the flow of events or interactions is defined as follows.

Flow of Events for Main Success Scenario:

1. →(a) Included UC-6: Login. (b) The administrator edits the records..
2. ← System (a) displays the delivery records chronologically in the displayer.

Flow of Events for Extensions (Alternate Scenarios):

1. ← System display “There are no delivery records for you” to the log-in user.
2. → The user contact with the administrator to find out the problem.
3. ← System forbid administrator accessing/editing the delivery record.
4. → The administrator reset the system.

The user story of ST2 require that user can check the delivery status of current delivery and the delivery history of the past deliveries and the story of ST9 require that administrator can check every delivery record of each user and edit the record if it is necessary, so we should decide that authenticated person can be access to delivery record.

Notify Error	
Related requirements	ST12 stated in Table 3-1.
Initiating actor	System.
Actor's goal	To notify the administrator and users there are errors occur in the delivery system.
Participating actors	Display, Email.
Preconditions	The delivery system meet some problems it cannot solve itself such as mechanical problems, missing ways.
Postconditions	

Table 4.9.: Summary for the use case UC-13: Notify Error

UC-2	TrackDeliveryStatus
Related User Stories	ST-10
Initiating Actor	Any of: Sender, Receiver
Actor's Goal	To check the delivery status of a package or mail
Preconditions	Sender/Receiver are registered user
Success End Condition	Sender/Receiver has access to his or her specific delivery status

Table 4.10.: Summary for the use case UC-2: TrackDeliveryStatus

4.7.8. Notify Error

Table 4.9 shows the details of the Notify Error use case, and the flow of events or interactions is defined as follows.

Flow of Events for Main Success Scenario:

1. → The delivery system meet some problems it cannot solve itself such as mechanical problems, missing ways.
2. ← System (a) sends the error notification to the administrator, (d) displays the error notification to any log-in users.

The user story of ST10 requires that administrator and user should be notified when there are some non-automatically solvable problems, so we should decide that administrator can receive notification email about the problem of the system and user can find the notification about the error after logging into the system.

4.7.9. Track Delivery Status

Table 4.10 shows the details of the Track Delivery Status use case, and the flow of events is defined as follows:

1. Include Login (UC-6)
2. ← System shows a screen containing a list of options, including "Check delivery status".
3. → Sender/ Receiver selects the option "Check delivery status".
4. ← System shows a screen with (i) a list of deliveries (including time, date, sender, receiver, type and current status) within n days (default n is 10, and it can be changed by user by using the selection box), (ii) a select box for user selecting a valid day number, (iii) a search box for user searching the specific delivery.
5. Sender/Receiver
 - a) → selects one delivery status

UC-11	Authentication to Robot
Related User Stories	ST-6, ST-9, ST-11
Initiating Actor	Any of: Sender, Receiver
Actor's Goal	To be authenticated by the robot and can open the mailbox in robot for delivery/receiving.
Participating Actors	LED, Administrator, Timer
Preconditions:	The set of valid keys stored in the system database is non-empty. The counter of authentication attempts equals zero. The LED is red indicating invalidity.
Success End Condition	Sender/Receiver successfully open the mailbox in the robot for delivery/receiving.
Failed End Condition	Sender/Receiver failed to open the box and a delivery failure will be reported. The robot stops this task and continues next task.

Table 4.11.: Summary for the use case UC-11: AuthenticationToRobot

- b) → changes the day number to any valid number
 - i. ← System goes to step 4.
- c) → searches using a word (such as name or delivery type, or a portion of it).
 - i. System
 - A. 1. ← finds and returns one or more than one results associate to the using word
 - 2. → Sender/Receiver selects one specific delivery status
 - B. 1. ← does not find any match, (a) returns to an error telling this and (b) display a continue botton.
 - 2. → Sender/Receiver clicks continue.
 - 3. ← System goes to step 4.
- 6. ← System shows a screen of detailed information about one specific delivery, including the package location and corresponding status at each specific time. And this display is continuously updated.

Comments Note that the main success scenario (check the delivery status of package or mails) is composed by:

1 2 3 4 5a 6

4.7.10. Authentication to Robot

Table 4.11 shows the details of the Authentication to Robot use case, and the flow of events or interactions is defined as follows:

1. ← System prompts the sender/receiver for identification, e.g., alphanumeric key
2. Sender/Receiver
 - a) → enters valid identification key
 - b) → enters an invalid identification key
 - i. ← System detects error, marks a failed attempt and increases the counter in the database.
 - ii. ← System check whether the counter of failed attempts exceeds the maximum allowed number
 - A. If yes, System (a) report unsuccessful delivery/receiving to administrator, and (c) take charge of the next delivery work.
 - B. If not, go to Step 1.

UC-12	NotifyReceiver
Related User Stories	ST-7
Initiating Actor	Timer
Actor's Goal	To notify receiver about the package or mail before arriving.
Participating Actors	Receiver
Preconditions	Receiver is a registered user.
Success End Condition	Receiver receives the notification email from system.

Table 4.12.: Summary for the use case UC-12: NotifyReceiver

UC-3	ObtainHelp
Related User Stories:	ST - 14
Initiating Actor:	User
Actor's Goal:	To obtain guidance about how this system works, and what a user need to do to finish the delivery process
Preconditions:	The user have access to the internet
End Conditions:	The user obtain orientation information about the system

Table 4.13.: Summary for the use case UC-3: ObtainHelp

3. ← System (a) verifies that the key is valid, (b) signals to the LED of the key's validity
4. ← System signals the Timer to start turn LED to invalidity timer countdown. 5. → Sender/Receiver opens the mailbox for sending/receiving mail or packages

Comments Note that the main success scenario (To be authenticated by the robot and can open the mailbox in robot for delivery/receiving.) is composed by:

1 2a 3 4 5

4.7.11. Notify Receiver

Table 4.12 shows the details of the Notify Receiver use case, and the flow of events is defined as follows:

1. ← System signals to the Timer to start the timer countdown at the beginning of the delivery.
2. → After a specific time, System sends a notification email to receiver automatically.
3. → Receiver receives the notification email

Comments Note that there is no failed end condition, because the preconditions specify that the receiver is a registered user in this system. For system simplicity, we defined that registered user will successfully received the notifying message if there is no system non-automatically error.

4.7.12. Obtain Help

Table 4.13 shows the details of the Obtain Help use case, and the flow of events is defined as follows:

1. ← System signals to the Timer to start the timer countdown at the beginning of the delivery.
2. → After a specific time, System sends a notification email to receiver automatically.
3. → Receiver receives the notification email

Flow of Events for Main Success Scenario:

1. → The user visit webpage of the system or open the mobile application of the system;
2. → The user click the "Help" link;

UC-7	PickUpPackage
Related User Stories:	ST-5, ST-9, ST-15
Initiating Actor:	Timer
Actor's Goal:	To drive the robot to the sender's office, request authentication to the sender, verify and pickup package from the sender
Participating Actors:	Sender
Preconditions:	1. The sender is registered in the autoDelivery System; 2. The sender has successfully booked a delivery in the system;
Success End Conditions:	The sender put the package in one compartment of the container in the robot, then the robot leave for delivery;
Fail End Conditions:	If any of the closed compartment of robot is opened during this process, an alert will be sent to the administrator;

Table 4.14.: Summary for the use case UC-7: PickUpPackage

- ← The help page is shown in the web page or mobile application

The help instruction includes: What is AutoDelivery System, How to book a delivery, How to register for an account, what to do when a robot is here to pick up your package, how to authenticate to the robot, how to receive a package etc.

4.7.13. Pickup Package

Table 4.14 shows the details of the Pickup Package use case, and the flow of events is defined as follows:

Flow of Events for Main Success Scenario:

- timer initiates the robot to move from current location to the sender's office.
- Include::MoveRobotToPoint(UC-15);
- Include::AuthenticationToRobot(UC-11);
- The sender put his or her package into the open compartment of container in robot;
- The sender acknowledges the system that the process of pick up is finished by pressing the button in the robot and the system store the status that the package has been picked up
- ← The mobile system(robot) leave for delivery to the receiver;

This use case facilitates the sender's delivery process, because the sender will not need to go to a centralized "post office" to mail his or her package. Instead, the mobile part of the system (robot) will come to the sender for a pick-up. The prerequisite of this use case is that the sender has already finished the "book a delivery" process in the system. The system will drive the robot to come to the sender's office, request for the sender to input password of his or her own account (the same account as when he or she log in to his account). The LED of a free compartment of the container (which is not in use currently) will turn green (as an indication that the cover is open). Here we use a LED with color of red or green in each compartment of the container to indicate the compartment to be locked or open, instead of using real lock equipment in order to simplify the design. Then the sender is able to put his or her package into the opened compartment. After that, the sender presses the button as a confirmation to the system that the pickup has finished. The robot will leave the sender's office and head to the receiver's office for a delivery.

For the possibility that the sender may not be available at the time the robot arrives at the sender's office, which includes several scenarios, will be implemented later.

4.7.14. Delivery

Table 4.15 shows the details of the Delivery use case, and the flow of events is defined as follows:

Flow of Events for Main Success Scenario:

UC-8	Delivery
Related User Stories:	ST-6, ST-7, ST-8, ST - 15
Initiating Actor:	Timer
Actor's Goal:	To notify receiver, drive the robot to receiver's location, authenticate and open the container to give package to the receiver
Participating Actors:	Receiver
Preconditions:	1. The sender booked a delivery from the system. 2. The Package has been successfully picked up from the sender by robot.
Success End Conditions:	The robot opens the compartment containing receiver's package. The receiver picked up the package and the robot leave.
Fail End Conditions:	1. If the receiver is not in his office, the robot will wait for one minute, mark as first false attempt and leave; 2. If false attempt is greater than three, the robot will return the package to the sender;

Table 4.15.: Summary for the use case UC-8: Delivery

1. → The timer initiates the mobile parts of the system to start the Delivery process;
2. ← The system send receiver an email to notify receiver about the package;
3. ← The system drives the robot to move to the receiver's office(Include:: MoveRobotToPoint(UC-15));
4. ← When the robot arrives at the location, the system wait for the receiver to response and in one min,
5. → the receiver comes to authenticate(Include::AuthenticationToRobot(UC-11));
6. →The receiver take out package from the open compartment;
7. → The receiver press the button as a confirm that delivery is finished;
8. ← The robot leave the receiver's office and continue the remaining task;

Flow of Events for Extensions(Alternate Scenarios):

- 5a. The receiver doesn't come to the robot because of unavailability in one min(first attempt or second attempt)
 1. ← The system set the "first/second attempt failure" flag to this delivery and leave for next task;
 2. ← The system will assign a second delivery to the person when there is no task waiting.
- 5b. The receiver doesn't come to the robot for the third delivery attempt to the receiver;
 1. ← The system will send an email to the sender about an return of the package;
 2. ← The system will move the package back to the sender's office.

During the delivery process, the system will first send a notification email to the receiver about the delivery. Then the system will driver the robot to the receiver's office and wait for the receiver to input his or her password (the same as the account password). The compartment that contains the package will not open until receiver inputs correct password of his or her account. Then the LED of the corresponding compartment will turn green, indicating that the compartment is open and ready for the receiver to take out the package. The receiver needs to press the confirm button on the robot when he has taken the package out of the compartment. The LED of this compartment will turn red as "locked" and the delivery process finishes. In case of failing to deliver the package to receiver, for example, the receiver is not in his or her office while the robot arrives, robot will wait for one minute, mark this trial as first attempt and leave for the next task. The robot will return the package to the sender if the system has tried three times to the receiver but still failed.

UC-4	ManageUser
Related User Stories:	ST-1
Initiating Actor:	Administrator or user
Actor's Goal:	Allows the administrator to view the user's information and provide optional choice of adding user, removing user and edit user information
Preconditions:	User is authorized as a Administrator or user and login in with valid account: include Login (UC-6)
Success End Conditions:	The Administrator can get access to user information interface in the system. The user can get access to his or her own information
Fail End Conditions:	The administrator fails to update the user information.

Table 4.16.: Summary for the use case UC-4: ManageUser

4.7.15. ManageUser

This flow specifies how a administrator or user can get access to the ManageUser interface (see table 4.16). Here is a extension point at the flow procedure 5. Although the administrator and user both will be shown the ManageUser interface that contains three option selection, these options somehow have the different responsibilities and authorities.

Flow of Events for Main Success Scenario:

1. → Include : Login
2. ← System displays main user interface (UI).
3. → Administrator/User selects "ManageUser" the system UI.
4. ← System displays a menu containing a list of user information and option selections: add user, remove user and edit user information.
5. → Administrator/User selects desired option.
 - a) Selects Add User, Include AddUser (UC-10).
 - b) Selects Edit User Information, Include EditUserInformation (UC-9).
 - c) Selects Remove User, Include RemoveUser (UC-19).
 - d) Selects Cancel
 - i. ← System goes back to Step 2
6. ← System records and updates the newest information

4.7.16. EditUserInformation

This use case illustrates the procedure of editing user information (see table 4.17). For the administrator, he or she has access to all user items and has the right to edit all user information items. For the user, he or she only has access to his or her own user information and has the right to update. After confirming edition, the new edited information will automatically be updated in the system database.

Flow of Events for Main Success Scenario:

1. →The administrator or user select the option ManageUser on the ManageUser interface
2. The system
 - a) ← display all users item for the administrator to edit.
 - b) ← display user himself or herself information for the user to edit.
3. Two options:

UC-9	EditUserInfo
Related User Stories:	ST-1
Initiating Actor:	Administrator or user
Actor's Goal:	Allows the administrator to edit the user's information Allows the user to edit his or her own information
Preconditions:	User is authorized as a Administrator or user and login in with valid account: include Login (UC-6)
Postconditions:	The Administrator can get access to user information interface in the system The user can get access to his or her information interface in the system.
Fail End Conditions:	The administrator or user fails to update the user information.

Table 4.17.: Summary for the use case UC-9: EditUserInfo

- a) → The administrator or user edit the desired text information
- b) → The administrator selects the confirm button option
4. ←The system validate the information and update.

4.8. Acceptance tests for Use Cases

Next we will describe the acceptance tests that the customer will need to run when the use case is implemented to convince him/herself that it is working.

Table 4.18 shows the details of the test case for the main success scenario of the BookDelivery use case. Alternate flows such as search by name (table 4.19), search by address (table 4.20), multiple delivery test (table 4.21) or timeframe test (table 4.22) are shown. Fail use case' tests are also described (see table 4.23).

Note that with these use case's tests we are not exhaustive at all, because, for example, some paths are not tested (e.g. multiple delivery searching by name). However, this will be tested deeply with unit and integration tests.

On the other hand, the test shown in Table 4.18 also includes the testing of the main scenario of the use cases UC-15 MoveRobotToPoint, UC-18 PositionInspection and UC-16/17 SetMotorSpeedL/R.

From the flow of events or interactions in Track Delivery Status use case, we can see that there are a main success scenario and several alternate scenarios. Here we fully test these four scenarios by providing four specific test cases. The table 4.25 shows the test case TC-2.1, which is used to test the main success scenario of UC-11 Authentication to Robot, and the main success scenario is user or sender successfully checks the delivery status of a package or mail.

Table 4.26 shows the details of test case TC-2.2, which is used to test an alternate scenario in UC-2 Track Delivery Status. The track delivery UI will automatically displays a menu containing 10 day's delivery information. If a user wants to change the default 10 days and check for the delivery information for specific days, he or she can change the day number to achieve this. System works similarly as the default set.

Table 4.27 shows the details of test case TC-2.3, which is used to test another alternate scenario in UC-2 Track Delivery Status. The user may want to check delivery information quickly through searching. If user searches a valid word, system will display a list delivery information associated with the valid word, and then the user can select the specific delivery for delivery status track.

Table 4.28 shows the details of test case TC-2.4, which is used to test another alternate scenario in UC-2 Track Delivery Status. If during the searching, user enters an invalid word, system will shows a menu saying that there is no delivery matches to that word. User can then click the Go Back button to return to the main menu.

TC-1.01	
Use case tested	UC-1 BookDelivery, main success scenario. UC-7 PickUpPackage. UC-15 MoveRobotToPoint. UC-18 PositionInspection. UC-16/17 SetMotorSpeedL/R.
Pass/fail criteria	The test passes if the sender books a single delivery, the system confirms that is stored in the database and the robot appears in front of the sender's office.
Input data	The user needs to choose a valid starred receiver and ASAP as a time frame.
Procedure	Expected result
Setup: A sender logs in correctly. Run test TC-1.01 in order to have one or more starred receivers.	The system show a list of options, possible with the BookDelivery as a default. The system shows a list of starred receivers (at least 1)
1. Sender selects the book delivery option from the menu (if this option does not appear in the front page).	The system shows a list of receivers.
2. Sender selects a receiver from the starred list.	The selected receiver should become now the receiver for the current booking. The system shows possible timeframes to pick up the package.
3. Sender selects the nearest time frame pick up (ASAP).	The user can book more deliveries.
4. Sender does not book multiple deliveries.	The system shows the selected receiver and pick up timeframe in the summary.
5. Sender confirms selection	The system should show a acknowledgement message, and the robot should appear in front of the office in the selected timeframe.
6. continue testing like in TC-1.01 (PickUpDelivery, main success scenario).	

Table 4.18.: TC-1.01 Test case for UC-1: BookDelivery

TC-1.02	
Use case tested	UC-1 BookDelivery, alternative scenario 5.b). UC-7 PickUpPackage.
Pass/fail criteria	The test passes if the sender books a single delivery, the system confirms that is stored in the database and the robot appears in front of the sender's office.
Input data	The user needs to search for a non-starred receiver using his/her name and ASAP as a time frame.
Procedure	Expected result
Exactly as in TC-1.01, except step 2 is substituted by:	
2.1 Sender searches an existing receiver's name.	The system shows a list of matches (in our case, our searched receiver must appear).
2.2 Sender chooses the searched receiver	The selected receiver should become now the receiver for the current booking. The system shows possible timeframes to pick up the package.

Table 4.19.: TC-1.02 Test case for UC-1: BookDelivery

TC-1.03	
Use case tested	UC-1 BookDelivery, alternative scenario 5.c). UC-7 PickUpPackage.
Pass/fail criteria	The test passes if the sender books a single delivery, the system confirms that is stored in the database and the robot appears in front of the sender's office.
Input data	The user needs to search for a non-starred receiver using his/her address and ASAP as a time frame.
Procedure	Expected result
Exactly as in TC-1.01, except step 2 is substituted by:	
2.1 Sender searches an existing receiver's address (building and office).	The system shows a list of matches (in our case, our searched receiver must appear).
2.2 Sender chooses the searched receiver.	The selected receiver should become now the receiver for the current booking. The system shows possible timeframes to pick up the package.

Table 4.20.: TC-1.03 Test case for UC-1: BookDelivery

TC-1.04	
Use case tested	UC-1 BookDelivery, alternative scenario 9.a). UC-7 PickUpPackage.
Pass/fail criteria	The test passes if the sender books multiple deliveries, the system confirms that is stored in the database and the robot appears in front of the sender's office.
Input data	The user needs to select several starred receiver and ASAP as a time frame.
Procedure	Expected result
Exactly as in TC-1.01, except that step 4 is substituted by:	
4.1 The user wants to book another delivery (up to 5)	The system returns to step 1.
4.2 Continue to step 1.	

Table 4.21.: TC-1.04 Test case for UC-1: BookDelivery

TC-1.05	
Use case tested	UC-1 BookDelivery, alternative scenario 7). UC-7 PickUpPackage.
Pass/fail criteria	The test passes if the sender books a single delivery, the system confirms that is stored in the database and the robot appears in front of the sender's office in the selected timeframe (with an error of ± 5 mins).
Input data	The user needs to search for a starred receiver and a time frame different than ASAP.
Procedure	Expected result
Exactly as in TC-1.01, except step 3 is substituted by:	
3. Sender selects a random time frame pick up different than ASAP.	The user can book more deliveries.

Table 4.22.: TC-1.05 Test case for UC-1: BookDelivery

TC-1.06	
Use case tested	UC-1 BookDelivery, error scenario 5.b.i.B).
Pass/fail criteria	The test passes if the sender looks for a non-existing user and the system confirms correctly handles the exception and shows the main screen.
Input data	The user needs to search for a non-existing receiver using his/her fake name.
Procedure	Expected result
Setup: A sender logs in correctly.	
1. Sender selects the book delivery option from the menu (if this option does not appear in the front page).	The system shows a list of receivers.
2. Sender searches for a fake name.	The system says that the name searched was not found.
3. Sender acknowledges the error message.	The system shows the expected output of step 1.

Table 4.23.: TC-1.06 Test case for UC-1: BookDelivery

TC-14.1	
Use case tested	UC-14 ChargeBattery, main success scenario.
Pass/fail criteria	The Test passes if a) The system is informed when the battery charge limit is under the first specified limit by BWM 1 and when the battery is about to finish by BWM 2. b) The system orders the current tasks and automatic battery charging task making sense to the designed strategy when it receives the BWM 1. c) The system goes into automatic battery charging mode (blocking the current tasks of the delivery machine and it goes to specified charging place) when it receives the BWM 2.
Input data	Actor BatteryLevelIndicator should send the BWM 1 and BWM 2 to the system correctly.
Procedure	Expected result
1. Setup: System is loaded by considerable number of delivery tasks and it begins to take action according to the tasks.	Delivery machine schedules and goes its way to deliver the assigned packets.
2. BatteryLevelIndicator sends the warning message, BWM 1, to the system.	System sends the corresponding directive to the delivery machine and delivery machine orders its current delivery tasks and automatic charging task based on the designed strategy. (This so-called “designed strategy” is a future work).
3. . BatteryLevelIndicator sends the warning message, BWM 2, to the system.	System sends the corresponding directive to the delivery machine and delivery machine blocks its current tasks and goes to specified automatic charging area and charge itself .

Table 4.24.: TC-14.1 Test case for UC-14: ChargeBattery

TC-2.1	
Pass/fail Criteria	The test passes if the user selects one specific delivery status, and system shows the detailed delivery information of that specific delivery.
Input Data	Clicking the Status Checking function in the web user interface (UI).
Procedure	Expected result
Set Up: Sender/Receiver is registered user, and has logged in the web user interface successfully by using the valid key.	
Step 1: User selects the Status Checking function in the web UI.	System shows a screen with a list of deliveries (including time, date, sender, receiver, type and current status) within 10 days.
Step 2: User selects one specific delivery status	System shows a screen of detailed information about the specific delivery, including the package location and corresponding status at each specific time.

Table 4.25.: TC-2.1 Test case for UC-2: TrackDeliveryStatus

TC-2.2	
Pass/fail Criteria:	UC-2 alternate scenario The test passes if the user selects the default day number to any valid number n, and selects one specific delivery status afterwards; system shows the detailed information of the specific delivery.
Input Data:	Clicking the Status Checking function in the web user interface (UI).
Procedure	Expected result
Set Up: Sender/Receiver is registered user, and has logged in the web user interface successfully by using the valid key.	
Step 1: User selects the Status Checking function in the web UI.	System shows a screen with a list of deliveries (including time, date, sender, receiver, type and current status) within 10 days.
Step 2: User changes the default day number to any valid number n	System shows a screen with a list of deliveries (including time, date, sender, receiver, type and current status) within n days.
Step 3: User selects one specific delivery status	System shows a screen of detailed information about the specific delivery, including the package location and corresponding status at each specific time.

Table 4.26.: TC-2.2 Test case for UC-2: TrackDeliveryStatus

TC-2.3	UC-2 alternate scenario
Pass/fail Criteria:	The test passes if the user searches using a valid word (such as a colleague's name or a portion of it), system shows the detailed information of the specific delivery.
Input Data:	Clicking the Status Checking function in the web user interface (UI).
Procedure	Expected result
Set Up: Sender/Receiver is registered user, and has logged in the web user interface successfully by using the valid key.	
Step 1: User selects the Status Checking function in the web UI.	System shows a screen with a list of deliveries (including time, date, sender, receiver, type and current status) within 10 days.
Step 2: User searches using a colleague's name or a portion of it	System shows a screen with a list of deliveries associating with the colleague's name.
Step 3: User selects one specific delivery status	System shows a screen of detailed information about the specific delivery, including the package location and corresponding status at each specific time.

Table 4.27.: TC-2.3 Test case for UC-2: TrackDeliveryStatus

TC-2.4	UC-2 alternate scenario
Pass/fail Criteria:	The test passes if the user searches using an invalid word (such as a fake name), system shows a not found message and a Go Back button.
Input Data:	Clicking the Status Checking function in the web user interface (UI).
Procedure	Expected result
Set Up: Sender/Receiver is registered user, and has logged in the web user interface successfully by using the valid key.	
Step 1: User selects the Status Checking function in the web UI.	System shows a screen with a list of deliveries (including time, date, sender, receiver, type and current status) within 10 days.
Step 2: User searches using a invalid word, such as a fake name	System shows a message saying that search not found and displays a Go Back button.
Step 3: User clicks on the GO Back button	Systems shows a screen just like that the result in Step 1.

Table 4.28.: TC-2.4 Test case for UC-2: TrackDeliveryStatus

TC-11.1	UC-11 main success scenario
Pass/fail Criteria:	The test passes if the user enters a valid key, and system confirms that the key is stored in the database. The LED indicate validity and the mailbox can be open for delivery or receiving.
Input Data:	Numerical numbers
Procedure	Expected result
Set Up: The counter of authentication attempts equals zero; the set of valid keys stored in the system database is non-empty	
Step 1. User types in the valid key	The LED turned green to indicate successful authentication. The mailbox can be open to delivery or receiving.

Table 4.29.: TC-11.1 Test case for UC-2: TrackDeliveryStatus

TC-11.2	UC-11 alternate scenario
Pass/fail Criteria:	The test passes if the user enters invalid keys within the number of maximum allowed attempts, the LED indicates invalidity, the system prompts user to enter again.
Input Data:	Numerical numbers
Procedure	Expected result
Set Up: The counter of authentication attempts equals zero; the set of valid keys stored in the system database is non-empty	
Step 1. User types in an invalid key	The LED kept red to indicate failure. System prompts user to type in again.
Step 2. User types in the valid key	The LED turned green to indicate successful authentication. The mailbox can be open to delivery or receiving.

Table 4.30.: TC-11.2 Test case for UC-2: TrackDeliveryStatus

Authentication to Robot. From the flow of events or interactions in Authentication to Robot use case, we can see that there are a main success scenario and two alternate scenarios. Here we fully test these three scenarios by providing three test cases. The table 4.29 shows the test case TC-11.1, which is used to test the main success scenario (sender and receiver are authenticated by the robot and can open the mailbox in robot for delivery/receiving) of UC-11 Authentication to Robot.

Table 4.30 shows the details of test case TC-11.2, which is used to test an alternate scenario in UC-11 Authentication to Robot. The alternate scenario tested is that if a user enters invalid keys, but within the maximum allowed failed attempts, system will detect invalidity and prompts for entering again. As long as the valid key is entered, the authentication is confirmed.

Table 4.31 shows the details of test case TC-11.3, which is used to test another alternate scenario in UC-11 Authentication to Robot. This scenario is that if a user continuously enters invalid keys exceeds the maximum allowed failed attempts, system will detect invalidity all along. For the last time an invalid key is entered, the robot will give up this task and leaves to continue other delivery.

Pickup Package. This use case includes one test case (see table 4.33). This test case tests whether the robot will arrive at the sender's office, "open" the compartment when the password is correct and leave when the sender finishes putting packages into the compartment and press the confirm button. As this

TC-11.3	UC-11 alternate scenario
Pass/fail Criteria:	The test passes if the user continuously enters invalid keys for n times (n equals to the maximum allowed attempts number), the LED indicates invalidity all along and the robot leaves for another task for the last time.
Input Data:	Numerical numbers
Procedure	Expected result
Set Up: The counter of authentication attempts equals zero; the set of valid keys stored in the system database is non-empty	
Step 1. User types in an invalid key	The LED kept red to indicate failure. System prompts user to type in again.
Step 2. User continuously types in invalid keys for k times (k equals to the maximum allowed attempts number minus 1)	Just like step 1, similar result happens for m times (m equals to the maximum allowed attempts number minus 2). For the last time, the LED kept red all the time and robot gives up this task and leaves to continue other tasks.

Table 4.31.: TC-11.3 Test case for UC-2: TrackDeliveryStatus

TC-3.01	UC-11 UC-3, main successful scenario
Pass/fail Criteria	The test passes if the help page can be opened
Input Data:	User press “help” link in the website or mobile application
Procedure	Expected result
Step 1. User (sender/receiver or anyone who is capable to visit the webpage or open the mobile application);	
Step 2. Choose the “Help” link; System show a “help” page containing: What is AutoDelivery System, How to book a delivery, How to register for an account, what to do when a robot is here to pick up your package, how to authenticate to the robot, how to receive a package etc.	There is only one test case for UC-3. The prerequisite is that the user (any person who is willing to get to know about this system) is able to connect to the Internet. After the user visit the webpage or open the mobile application of this system, he or she will notice a “help” link in the screen. By clicking the link the user will be able to obtain help about knowing how this system works.

Table 4.32.: TC-3.01 Test case for UC-3: ObtainHelp

TC-7.01	UC-7, main successful scenario
Pass/fail Criteria	The test passes if the sender successfully put the package into one compartment of the robot's container and the robot leaves for delivery.
Input Data:	Book Delivery is complete
Procedure	Expected result
Step 1. Book delivery is complete;	The system will drive the robot from current location to the sender's place after the sender booked a delivery in the system;
Step 2. Wait for robot to come to the office	
Step 3. The sender inputs password of the account;	The LEDs will turn green when sender inputs correct account password as an indication that the box is open;
Step 4. Wait for the box to open after typing correct password;	
Step 5. Put in the package in the box;	
Step 6. Press the button as a confirm of finishing pickup process.	The LED will turn red and the system will drive the robot leave the sender's location for the receiver's location after the sender press the confirm button.

Table 4.33.: TC-7.01 Test case for UC-7: PickupPackage

use case includes two other use cases (UC-15 MoveRobotToPoint and UC-11 AuthenticationToRobot), the test case mainly tests whether the system will mark the status as “pickup Package” finishes and goes for the next step –Delivery, after the sender has pressed the button for confirmation.

This test case (see table 4.34) tests the successful scenario of UC-8 - receiver is available to receive the package. During this process, the receiver will first receive a notification email, and type in password when the robot comes. After LED of the corresponding compartment turns green, he or she is able to take the package out of the compartment and press the button in the robot as a confirmation.

This test case tests (see table 4.35) the condition that the receiver is not available at the first time or the second time the system is going to delivery the package to the receiver. After Pickup Package from the sender, the system will send a notification email to the receiver about delivering this package. If at the time the receiver is not available or not in his or her office, the system will make the robot wait at the receiver's office for one minutes. If in this one minute, the receiver comes back or is available, and start to input the password, the robot will continue to operate as a successful scenario; If the receiver does not respond to the system in this one minute, the robot will leave for the next delivery task (if possible) or wait at default position (if idle) and try delivering the package to the receiver for a 2nd time after 30min. If the 2nd time delivery attempt still fails, the robot will try a 3rd time.

This test case (see table 4.36) tests the condition that the receiver is not available when the robot arrives at the receiver's office for the 3rd time. The robot will arrive at the receiver's office, and wait for one minute. If during this time period, the receiver starts to enter his or her password, then the robot will verify the password, and turn LED to green if password is correct. Then the receiver can take the package from the compartment and press the button for confirmation as the delivery finishes. This test case mainly tests the condition that the user is still not available at this time. The robot wait for 1 min and the system will send a notification to the sender about returning this package, since the system regard this case as “unable to delivery”. The system then will drive the robot to the sender's place to return the package.

Table 4.37 is the acceptance test for the user case ManageUser. After any authorized user login in with valid account and password, they are prompted to the main interface of the ManageUser. As the priority difference, the administrator can get access to all user's information while the users can manage their own account. Here is a extension point from the ManageUser of three sub user case that to AddUser,

TC-8.01	UC-8, main successful scenario
Pass/fail Criteria	The robot successfully give the package to the corresponding receiver and leave receiver's location;
Input Data:	Pickup Package is complete
Procedure	Expected result
Step 1.Pickup Package is complete.	The system will send an notification email to the receiver after Pickup Package is complete;
Step 2.The receiver receive the email notification about the package.	The system will drive the robot to the receiver's location;
Step 3.The receiver wait for the robot to come;	The LED within the compartment that contains the receiver's package will turn green after the receiver has input the correct password;
Step 4.The receiver inputs password by the keypad of the robot;	
Step 5.The receiver take out the package from the compartment that has a LED with green light.	
Step 6.The receiver press the button in the robot as a confirm of finishing Delivery process.	The robot will leave for the next possible task after the receiver press the confirm button;

Table 4.34.: TC-8.01 Test case for UC-8: Delivery

TC-8.02	UC-8, alternate scenario (5a.)
Pass/fail Criteria	The robot leaves the receiver's office with his or her package safely stored in previous compartment in the case that the receiver is not in the office for the first or second time.
Input Data:	Pickup Package is complete; The robot comes to the receiver's location for a first or second attempt to delivery.
Procedure	Expected result
Step 1. Pickup Package is complete.	The system will send an notification email to the receiver after Pickup Package is complete;
Step 2. The receiver receives the email notification about the package.	The system will drive the robot to the receiver's location (at first or second attempt to delivery);
Step 3. The receiver is not in his or her office or are not available at the moment robot arrives (for the 1st or 2nd time to delivery).	The robot will wait at the receiver's office for 1 min for the receiver to take the package; The robot will leave for a next attempt to delivery (after finishing another pending task or after 30 min) if the 1 min time slot has expired;

Table 4.35.: TC-8.02 Test case for UC-8: Delivery

TC-8.03	UC-8, alternate scenario (5b.)
Pass/fail Criteria	The robot leaves the receiver's office with his or her package safely stored in previous compartment in the case that the receiver is not in the office for the 3rd delivery attempt and the robot will return the package back to the sender.
Input Data:	Pickup Package is complete; The robot has finished two delivery attempts to the receiver but both failed.
Procedure	Expected result
Step 1. Pickup Package is complete.	The system will send a notification email to the receiver after Pickup Package is complete;
Step 2. The receiver receives the email notification about the package.	The system will drive the robot to the receiver's location (at 3rd attempt to delivery when both 1st and 2nd attempt both failed);
Step 3. The receiver is not in his or her office or is not available when the robot tries to delivery the package to the receiver for the 3rd time.	The robot will wait at the receiver's office for 1 min for the receiver to take the package; After 1 min time expires, the system will send a notification email to the sender about returning this package; The system drives the robot to the sender's office and return the package;

Table 4.36.: TC-8.03 Test case for UC-8: Delivery

RemoveUser and EditUserInformation. As the three user cases are logically the same, we only choose the click on EditUserInformation button to finish the test and leave the other two out.

Tables 4.38, 4.39 and 4.40 are the acceptance test for the optional sub user case EditUserInformation, which extends from the Manage User. For the administrator, as the system identification in the ManageUser, he has the priority to edit every user information. As for the users in the second table, they only have the authority to edit their own information. For the last table, both the administrator and user can choose cancel edit and go back to the main interface of ManageUser if they don't want to edit the information. As for the acceptance test of user case RomoveUser and AddUser, the procedure,their relative priority difference and corresponding system response are somehow quite similar to the user EditUserInformation, so we will not further list and explain them all in the following.

TC-4.01	UC-4, main successful scenario with 5b
Pass/fail Criteria	The test passes if the administrator or user successfully modifies user account information into the system
Input Data:	The person is a registered user or administrator
Procedure	Expected result
Step 1. The registered user or administrator logs in successfully;	The system will display the manage user interface to the user;
Step 2. The user/administrator clicks the “Manage User” link;	The system will display the “Edit User Information” Interface to the user;
Step 3. The user/administrator edits user information	

Table 4.37.: TC-4.01 Test case for UC-4: ManageUser

TC-9.01	UC-9, main successful scenario
Pass/fail Criteria	The test passes if the administrator modify the information in the EditUserInformation interface and the system validates and update the information.
Input Data:	A user needs to edit user information
Procedure	Expected result
Step 1 The administrator clicks the EditUserInformation button;	The system will display the EditUserInformation window to the user;
Step 2 The administrator clicks on one user.	The system will display the interface of this user
Step 3 The administrator inputs the new information and click on the confirm button.	The system validates the information and update it to the database.

Table 4.38.: TC-9.01 Test case for UC-9: EditUserInformation for administrator

TC-9.02	UC-9, alternate scenario
Pass/fail Criteria	The test passes if the user modify his or her information in the EditUserInformation interface and the system validates and update the information.
Input Data:	A user needs to edit user information
Procedure	Expected result
Step 1 The administrator clicks the EditUserInformation button;	The system will display the EditUserInformation window to the user;
Step 2 The administrator clicks on his or her own user.	The system will display the interface of this user.;
Step 3 The administrator inputs the new information and click on the confirm button.	The system validates the information and update it to the database.

Table 4.39.: TC-9.02 Test case for UC-9: EditUserInformation for user

TC-9.03	UC-9, alternate scenario
Pass/fail Criteria	The test passes if the administrator or user modify the information in the EditUserInformation interface and the system validates and update the information.
Input Data:	A user needs to edit user information
Procedure	Expected result
Step 1 The administrator or clicks the EditUser-Information button;	The system will display the EditUserInformation window to the user;
Step 2 The administrator or user clicks on the cancel button.	The system will close of the EditUserInformation window and display the interface of the MangeUser.;

Table 4.40.: TC-9.03 Test case for UC-9: EditUserInformation for administrator or user

TC-6.01	
Use case tested	UC-6 Login, main success scenario 1.
Pass/fail criteria	The test passes if the user enters a name/password that is contained in the database and successfully log into the system as a user.
Input data	Correct user name and password.
Procedure	Expected result
1. Type in a registered user name	The user name displays in the login dialog box.
2. Type in a correct password	System prompt out a dialog box shows the login is successful with “Welcome, User XXX”; the user successfully access to the database of the system.

Table 4.41.: TC-6.01 Test case for UC-6: Login

TC-6.02	
Use case tested	UC-6 Login, main success scenario 2.
Pass/fail criteria	The test passes if the user enters “admin” and a special code and successfully log into the system as an administrator.
Input data	Admin, special code for administrator.
Procedure	Expected result
1. Type in “admin”	The “admin” displays in the login dialog box.
2. Type in a special code for administrator	System prompt out a dialog box shows the login is successful with “Welcome, administrator”; the administrator successfully access to the database of the system.

Table 4.42.: TC-6.02 Test case for UC-6: Login

TC-6.03	
Use case tested	UC-6 Login, alternate scenario.
Pass/fail criteria	The test failures if (a) the user enters wrong user name and/or password with less than three times and be notified the error and input again; (b) the user enters wrong user name and/or password three times and found the input dialog box is blocked.
Input data	Incorrect user name and password
Procedure	Expected result
1. Type in a unregistered user name.	The user name displays in the login dialog box.
2. Type in a password for another registered user.	System prompt out a dialog box shows the login is unsuccessful with “Sorry, the user name/password you entered is wrong, please enter again”.
3. Type in a registered user name.	The user name displays in the login dialog box.
4. Type in an incorrect password.	System prompt out a dialog box shows the login is unsuccessful with “Sorry, the user name/password you entered is wrong, please enter again”.
5. Type in a unregistered user name.	The user name displays in the login dialog box.
6. Type in a password randomly.	System prompt out a dialog box shows the login is unsuccessful with “You have entered the wrong user name/password three times and have been forbidden to log into our system today. We are sorry for any inconvenience it brings to you”.

Table 4.43.: TC-6.03 Test case for UC-6: Login

TC-5.01	
Use case tested	UC-5 Access Delivery Record, main success scenario 1.
Pass/fail criteria	The test passes if the user can find the delivery records stored in the system database after logging into the system.
Input data	Correct user name and password.
Procedure	Expected result
1. Log into the system as user	Access to the user account and find the delivery records display chronologically.

Table 4.44.: TC-5.01 Test case for UC-5: Access Delivery Record

TC-5.02	
Use case tested	UC-5 Access Delivery Record, main success scenario 2.
Pass/fail criteria	The test passes if the administrator can find and edit the delivery records stored in the system database after logging into the system as an administrator.
Input data	Correct user name and password.
Procedure	Expected result
1. Log into the system as administrator.	Access to the data base .
2. Step 2. Enter a user name.	All delivery records of this user list chronologically.
3. Delete a delivery record from the user account .	The selected delivery record of this user is deleted from the record list.

Table 4.45.: TC-5.02 Test case for UC-5: Access Delivery Record

TC-5.03	
Use case tested	UC-5 Access Delivery Record, alternate scenario 1.
Pass/fail criteria	The test failures if the user cannot his delivery records stored in the system database.
Input data	User name and password.
Procedure	Expected result
1. Log into the system as user.	Access to the data base and the system display "There are no delivery records for you" .

Table 4.46.: TC-5.03 Test case for UC-5: Access Delivery Record

TC-5.04	
Use case tested	UC-5 Access Delivery Record, alternate scenario 2.
Pass/fail criteria	The test failures if the administrator cannot access the delivery records of the users.
Input data	Admin and special code for administrator.
Procedure	Expected result
1. Log into the system as administrator.	Access to the data base .
2. Enter a user name.	There is no reaction of the system.
3. Repeat step 2 several times by using different user name.	There is no reaction of the system.

Table 4.47.: TC-5.04 Test case for UC-5: Access Delivery Record

TC-5.05	
Use case tested	UC-5 Access Delivery Record, alternate scenario 3.
Pass/fail criteria	The test failures if the administrator cannot edit the delivery records of the users.
Input data	Admin and special code for administrator.
Procedure	Expected result
1. Log into the system as administrator.	Access to the data base.
2. Enter a user name.	All delivery records of this user list chronologically.
3. Delete a delivery record from the user account.	There is no reaction of the system.
4. Repeat step 3 several times by using different user name.	There is no reaction of the system.

Table 4.48.: TC-5.05 Test case for UC-5: Access Delivery Record

TC-13.01	
Use case tested	UC-13 Notify Error, Main success scenario.
Pass/fail criteria	The test passes if (a) the administrator can receive notification email about the problem; (b) the user can find the error notification by logging into the system.
Input data	Admin and special code for administrator.
Procedure	Expected result
1. Taking apart the delivery system	The delivery system becomes several parts.
1. Booking a delivery	The administrator can receive notification email about the problem .
1. Logging into the system as a user.	The error notification display.

Table 4.49.: TC-13.01 Test case for UC-13: Notify Error

4.9. System sequence diagrams

Figure 4.2 shows a complete system sequence diagram for the BookDelivery use case. Specifically, it describes the main success scenario of the use case.

Figure 4.3 shows the equivalent diagram for the MoveRobotToPoint use case. Note, however, that it describes also the functionality of use cases SetMotorSpeed and PositionInspection (which is very simple). For space reasons the Actors MagneticSensorR and ReflectiveOpticalSensorR have been eliminated (behaves the same as MagneticSensorL and ReflectiveOpticalSensorL, respectively).

For UC-2 Track Delivery Status use case, one system sequence diagram representing the main success scenario is shown in the Figure 4.5. Note that in this diagram, loop iteration, which delineates a fragment that may execute multiple times, is used. Besides, alternative selection (such as select one delivery and searches one delivery) is also drawn in this diagram.

For UC-11 Authentication to Robot use case, two system sequence diagrams representing the main success scenario and another alternate scenario (burglary attempt scenario) are drawn in the Figures 4.6 and 4.7, respectively. Figure 4.7 shows “UML” loop iterations, which delineates a fragment that may execute multiple times.

4.10. Risk management

We detected a possible point of attack in BookDelivery use case. We solved it by placing a Business Policy: “one sender can only have 50 non-served book deliveries”.

We also detected a more general point of failure of the system: what if the system loose its path and can't recover it? Our solution is to send an e-mail to the administrator.

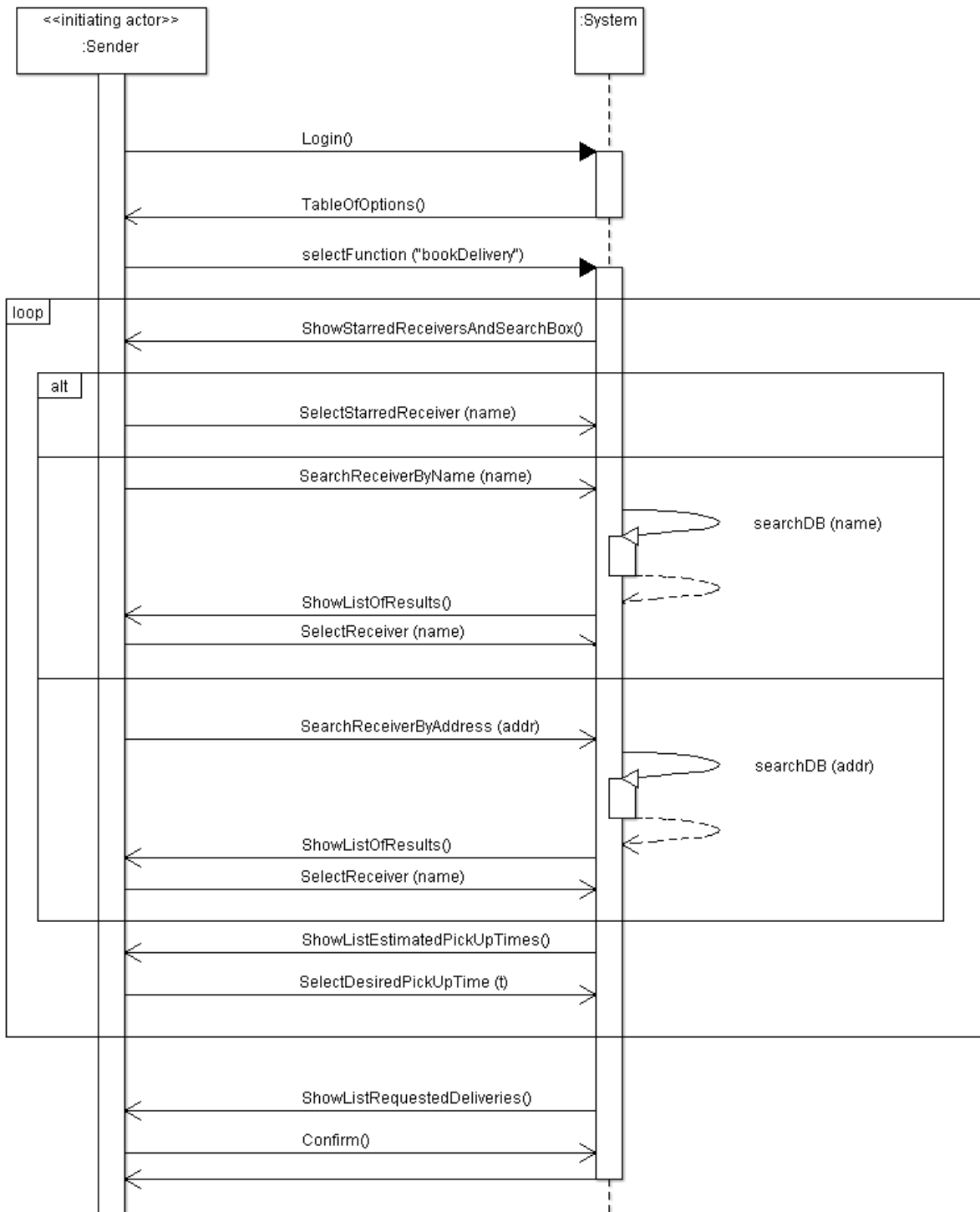


Figure 4.2.: System sequence diagram for the main usage case of the BookDelivery use case.

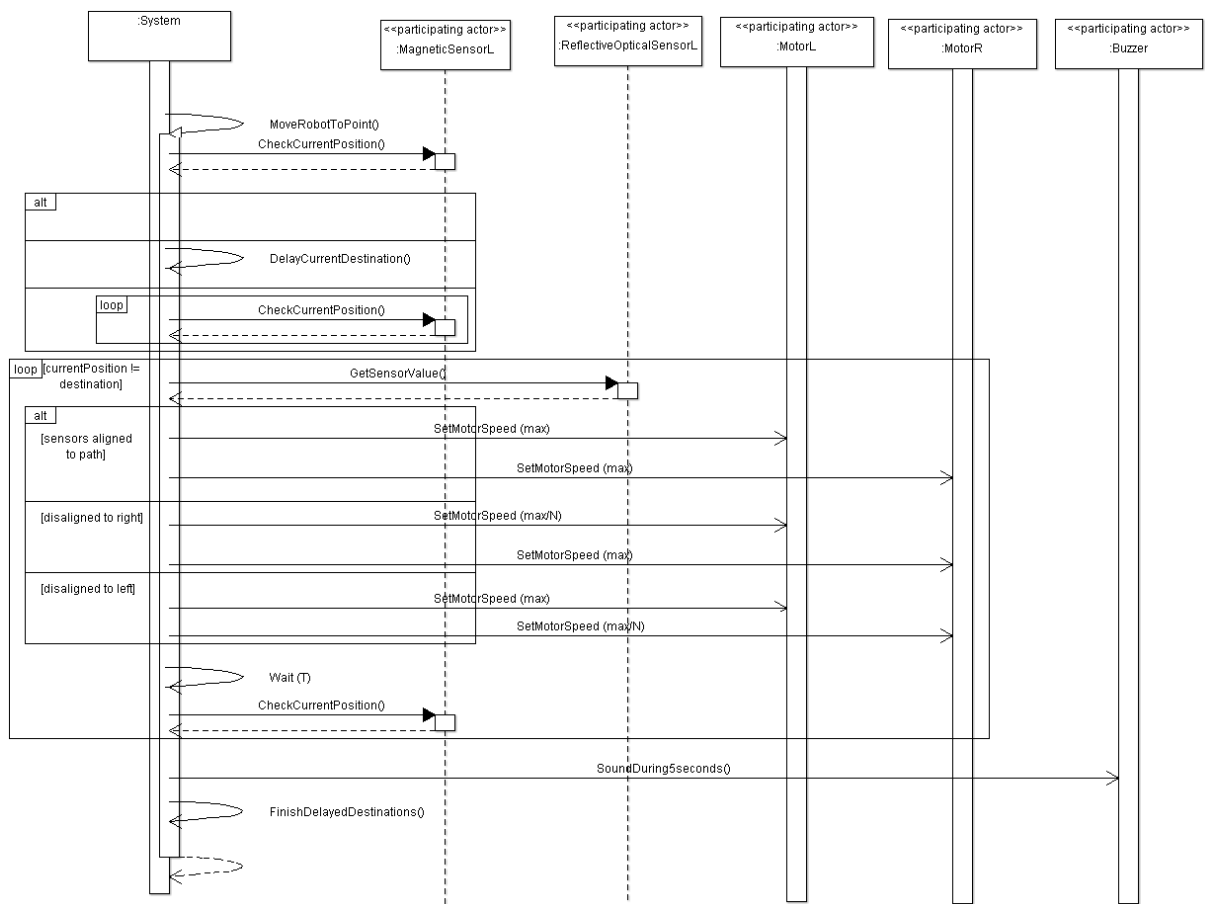


Figure 4.3.: System sequence diagram for the main usage case of the MoveRobotToPoint use case.

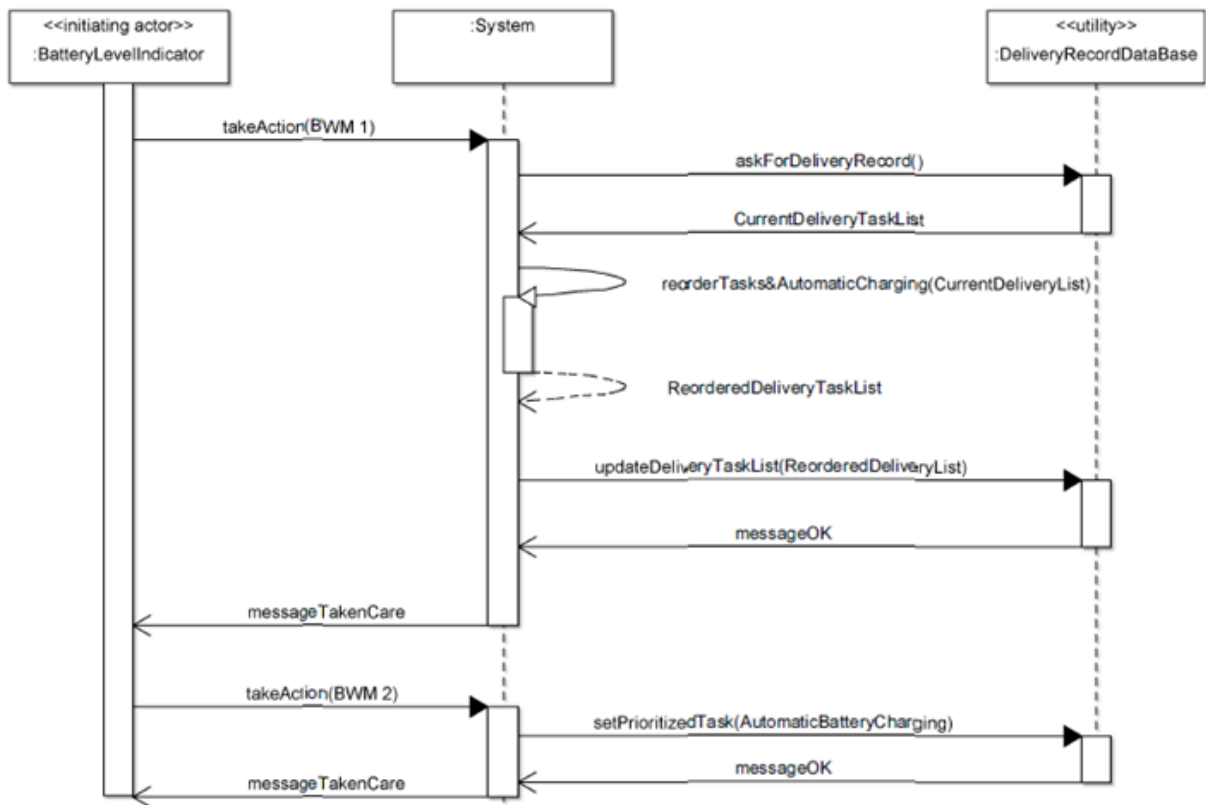


Figure 4.4.: System Sequence diagram for the casual case of ChargeBattery use case

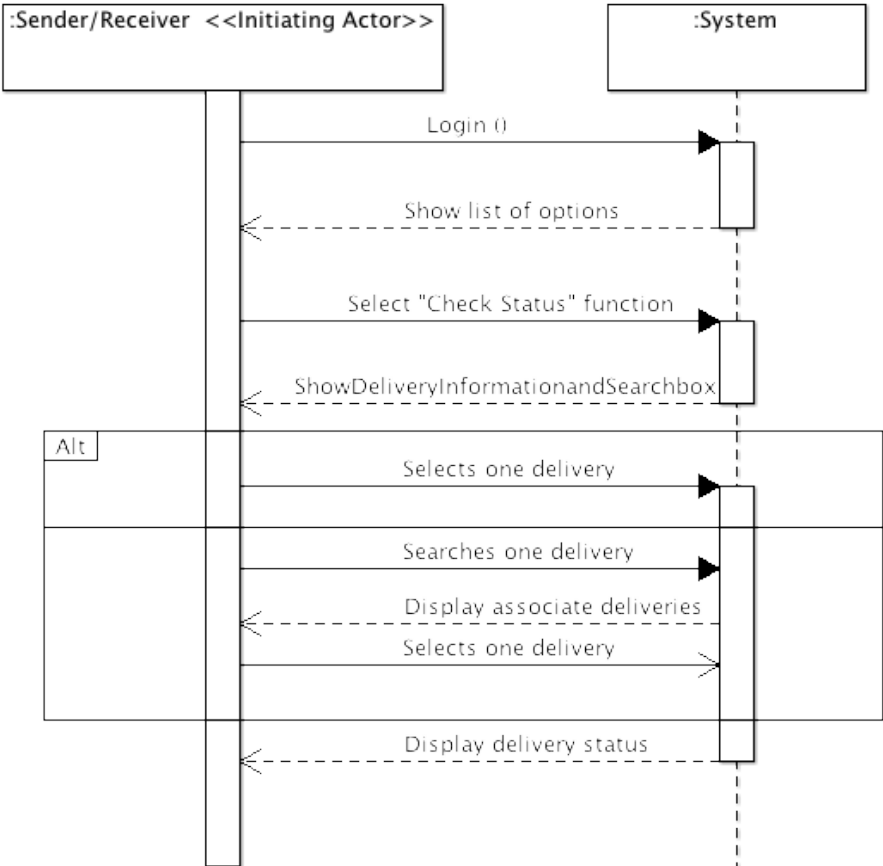


Figure 4.5.: UML System sequence diagrams for the main success scenario of UC-2 Track Delivery Status

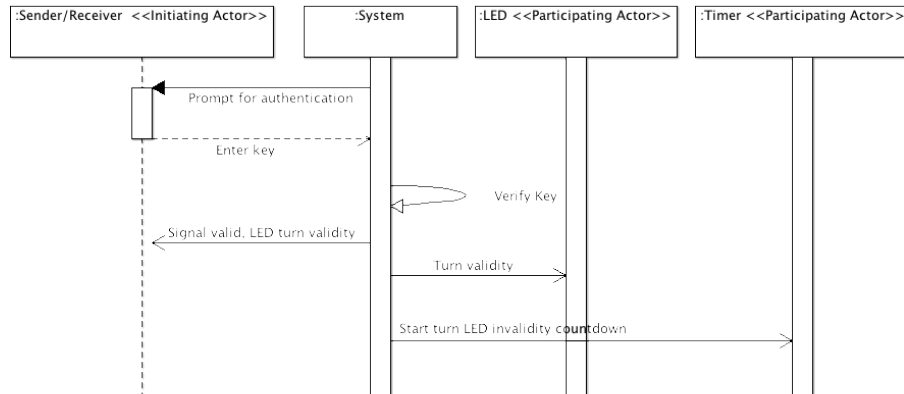


Figure 4.6.: UML System sequence diagrams for the main success scenario of UC-11

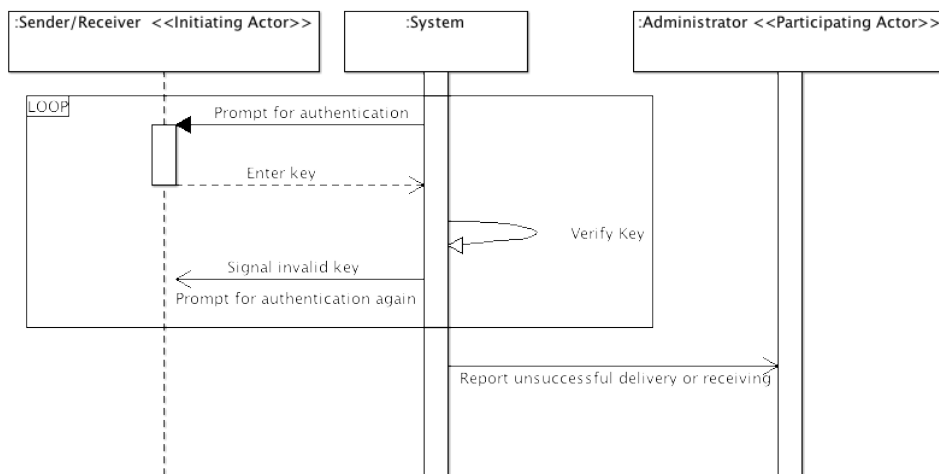


Figure 4.7.: UML System sequence diagrams for the alternate scenario of UC-11

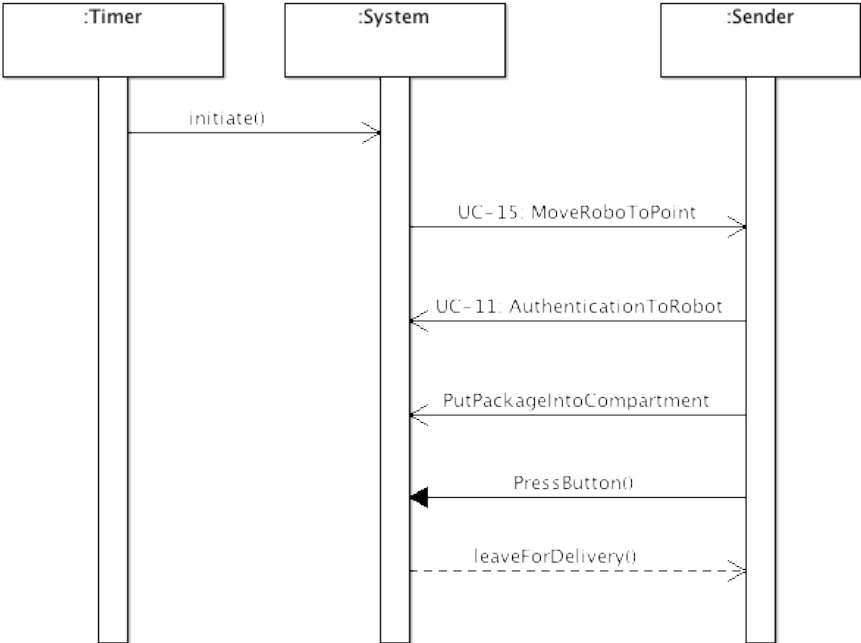


Figure 4.8.: Sequence diagram for main success scenario of package pick up mechanism, UC-7

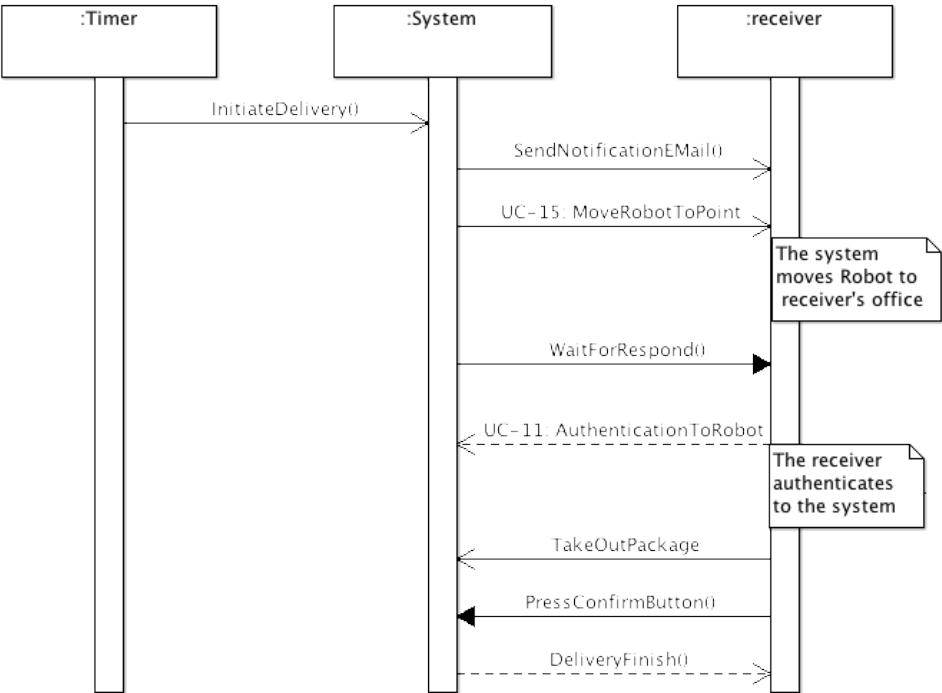


Figure 4.9.: Sequence diagram for main success delivery, UC-8

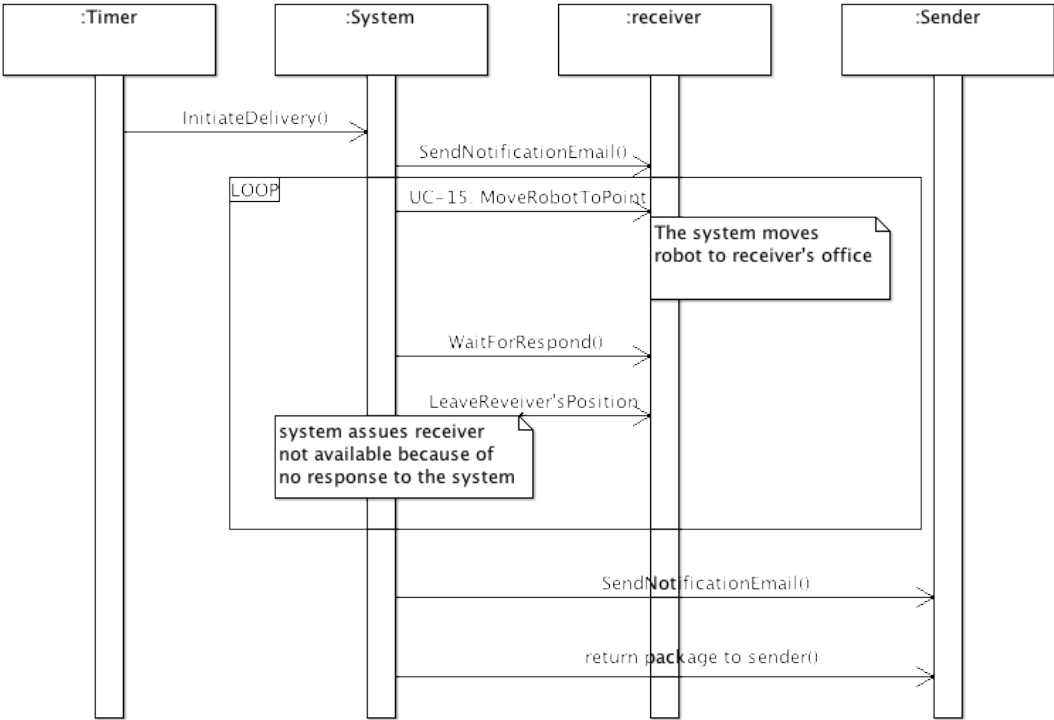


Figure 4.10.: Sequence diagram for failure delivery due to maximum number of attempts, UC-8

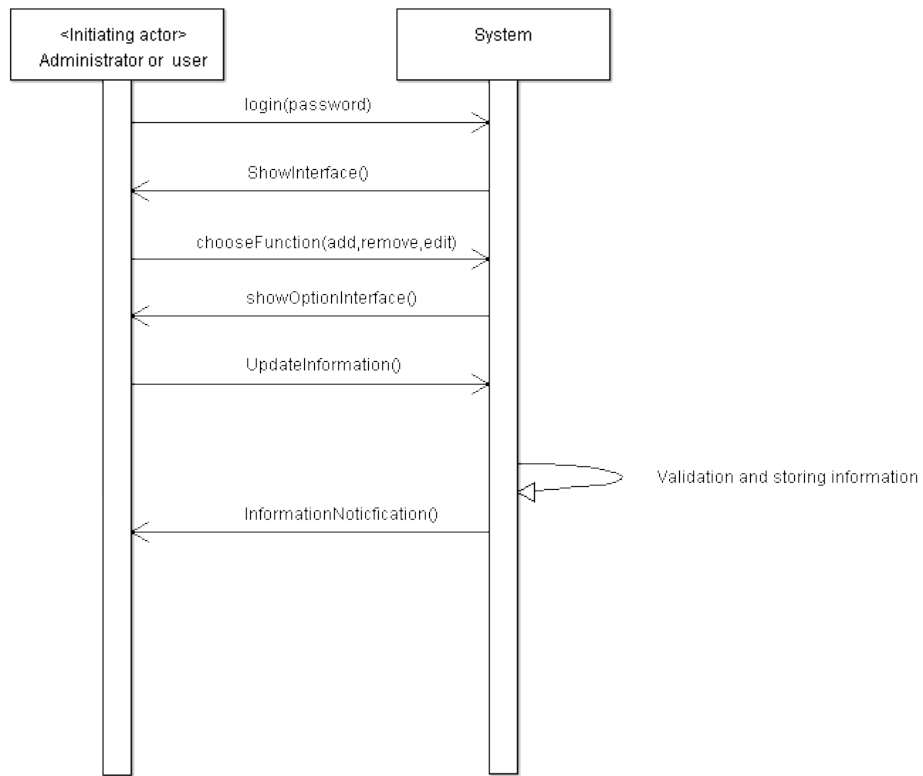


Figure 4.11.: Sequence Diagram for ManageUser use case, main success scenario

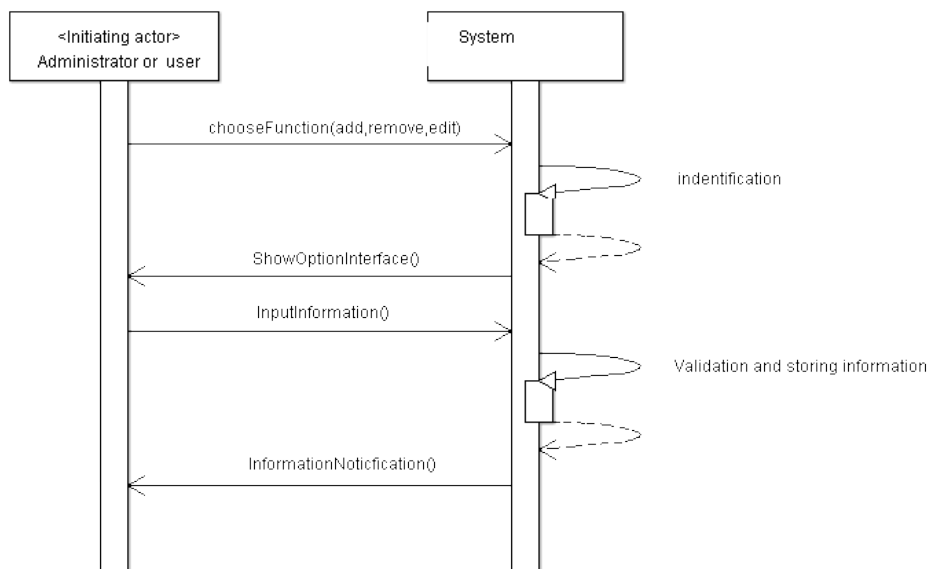


Figure 4.12.: Sequence diagram for Add, Edit or Remove user use cases, main success scenario

5. EFFORT ESTIMATION USING USE CASE POINTS

As Professor's book suggests in Section 4.6.1, use case points are one approach to measure the software size. Also, we need to know the teams' velocity (or productivity), which means how fast the team make progress in the design. Thus, we can use the following equation to derive the project duration:

$$DURATION = UCP * PF \quad (5.1)$$

Note that the productivity point here is the individual development hours per use case points. We assume the PF=28 for calculation simpleness. For the USE Case Points (UCP), it's a method to estimate the person-hours based on the corresponding user cases. Here is the equation:

$$UCP = UUCP * TCF * ECF \quad (5.2)$$

PTS	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12	UC13	UC14	UC15	UC16	UC17	UC18	UC19
UUCP	1.5	1	1	1	1	1	1.5	1.5	1	1	1	1	1	1	2	2	2	1	1
TCF	0.9	0.8	0.7	0.7	0.8	0.7	0.9	0.9	0.7	0.7	0.7	0.8	0.8	0.7	0.7	1	1	0.7	0.7
ECF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
UCP	1.35	0.8	0.7	0.7	0.8	0.7	1.35	1.35	0.7	0.7	0.7	0.8	0.8	0.7	1.4	2	2	0.7	0.7
PF	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28

Table 5.1.: Unadjusted Use Case Points, Technical Complexity Factor, Environment Complexity Factor and Productivity for UC1 to UC18

All the multiple factors are shown in above table. They are Unadjusted Use Case Points(UUCP), Technical Complexity Factor(TCF), Environment Complexity Factor (ECF)and Productivity Factor(PF) namely.

The UUCP measures the complexity of the functional requirement. In the 19 use cases of our project, the most important and complex 3 ones are UC15: MoveRobotToPoint, UC16:SetMotorSpeed and UC17: PositionInspection. They realize the main functionality of the our delivery system. Therefore, we give them the factor value of 2. The second most complex important user cases are UC11: Authentication-ToRobot, UC7: PickupPackage and UC8: Delivery. They are highly related to the delivery process and thus we give them the factor value of 1.5. Other use cases such as Obtainhelp, ManageUser are relatively easy to realize and has less complexity. According to these, they are assigned as 1. Next is the Technical Complexity Factor (TCF), it measures the complexity of the nonfunctional requirement. According to our project complexity, we assign the factor value between 0.5 to 1. Identically to the UUCP assignment method, we give the hardest 3 ones the value 1 and give the rest the value 0.8 and 0.7, which based on their nonfunctional complexity. For the Environment Complexity Factor (ECF), we estimate each group member's experience level and available time, we assign the value 1 to each of the use cases. Note that we consider the group members' experience and available time differences and then get the average. For the productivity factor, we assume it's 28 person-hours per use case point, as the requirement assumes. By using the equation above, we can obtain the sum of duration:

$$DURATION = \sum_{i=1}^{19} (UCP)_i * PF_i \quad (5.3)$$

After calculating, the sum of duration of 19 use cases are 19.95*28, which equals 560 hours. Note that this 560 hours include the things such as answering the emails, attending the meeting, writing the report,

preparing for the demo and so on. We assume we work 50 hours per week, which means we will complete our project in $560/50 = 11$ weeks. As our team has 6 members, we have totally 3360 hours for the whole design cycle.

6. DOMAIN ANALYSIS

6.1. Domain Model

To build the domain model we will revisit the fully-dressed description of the use cases, because we will be able to find or derive from there all the important concepts about the system. After this we will assign the required attributes and associations.

6.1.1. Concept definitions

6.1.1.1. Boundary concepts

Specifically, we will begin by analysing the boundary concepts. Therefore, we will begin analyzing the actors and their interactions with the system. As stated before, there are two kinds of actors: humans and non-humans. All the non-human actors (excepting the timer) are sensors or actuators in the robot, so we can deduce a first set of responsibilities about accessing/controlling them (and a possible concept to associate with):

- R1: operate the motors (MotorsOperator)
- R2: operate the buzzer (BuzzerOperator)
- R3: operate the status LEDs (StatusLEDsOperator)
- R4: access at the value of the reflective optical sensors (ReflectiveOpticalSensorsAccessor)
- R5: access at the value of the magnetic sensors (MagneticSensorsAccessor)
- R6: access at the level of battery charge (BatteryLevelIndicatorAccessor)
- R7: access at the value of the pressed keys in the numerical keyboard (NumericalKeyboardAccessor)

Note that we could have splitted more some concepts: for example, instead of including MotorsOperator we could be used MotorLOperator, MotorROperator (like we did in the use cases). However, we think that we can assign the task of controlling both motors' speed to the same 'worker' or to different workers that does exactly the same. The same can be applied to several other concepts.

Note also that is difficult to classify the role of the concepts that access sensors into a 'Do' or 'Know' something. However, as the mission of the sensors is to know some issue, the goal of the concept that access to it is purely a communication role: they are an interface. Therefore, we can classify them as 'Interface'-type concepts, as the book implies.

There is no such a direct way to identify boundary concepts related to human actors. Therefore, we will begin by detecting the responsibilities stated in the use case discussion:

- R8: display the information to the user via his/her computer
- R9: communicate several things to the user (such as which section of the secure box should he/she use, or failure entering the password) via the robot.
- R10: allow the user to interact with the system via his/her computer
- R11: allow the user to type the password and confirm pickup/delivery in the robot.
- R21: Send an e-mail to the receiver to notify a delivery.
- R22: Send an e-mail to the administrator, to notify an error.

As we can see R8 and R9 are related to the communication to the user and R10 and R11 are aimed to acquire information from the user.

Let's now map all the responsibilities into real concepts. One approach is to group them all in the sets **Interface** $\{R9, R11, R21, R22\}$ and **InterfaceRobot** $\{R1, R2, R3, R4, R5, R6, R7, R8, R10\}$. About the Interface concept, we think that model perfectly the reality, excepting the fact that the interface should not handle with e-mails. Therefore, we decide to separate in **Interface** $\{R9, R11\}$ and **EmailNotificator** $\{R21, R22\}$. However, although we think that InterfaceRobot may be too broad to model the domain clearly, considering one concept for each responsibility would end with too many very-simple concepts. That is why we decided to keep only the three broad-based concepts to handle with boundaries: Interface, EmailNotificator and InterfaceRobot.

6.1.1.2. Internal concepts

Now that we defined the boundary concepts, let's focus on the internal part of our problem domain.

'Know' concepts

To ease the internal concept identification we will try to find first which responsibilities there are that can be mapped to a 'know'-like concept. Again, we will continue the analysis of the discussed use cases to obtain these responsibilities:

- R12: Keep the list of users registered in the system with all the associated information with the account.
- R13: Keep the record of every delivery done, with all of its details.
- R14: Keep record of the mapping between the location marks and the corresponding office.
- R15: Keep record of pending deliveries, with its own priority.
- R16: Keep record of robot' current and last positions.
- R17: Keep record of recent state of the sensors (order of ms or s), in order to avoid noise. Also implement Business Policy ADS-BP06.
- R18: Keep information about the status of track detection.
- R27: Keep information about starred contacts for each user (business policy ADS-BP01).
- R28: Implement the limit on business policy ADS-BP03 (one sender can only have 50 non-served book deliveries).
- R37: Determine which section of the secure box is empty, which contains what, and which should be allocated for a new delivery.

Every requirement have the same goal: keeping information. However, each of them specifies a very differentiated set of data. Therefore, we think that every requirement here should be directly mapped to one single concept, obtaining: UserList (R12), DeliveryHistory (R13, R27), FloorMap¹ (R14), PendingDeliveries (R15, R28), RobotPosition (R16), Filter (R17), SecureBoxContents (R37).

However, we think that R18 is more suitable to be an attribute of the controller of track detection, that we will probably define later. On the contrary, we think that R16 should be a standalone concept, because it is not only related of path resolution, but also to provide hint to the administrator in case the robot gets lost. A similar argument can be apparent for the R17 requirement: we could map in an attribute of the boundary InterfaceRobot concept. However, we think that it might be a separate concept because it's role does not adjust adequately to the one in InterfaceRobot. R27 will be assigned to the 'know' concept DeliveryHistory, which actually have the knowledge about which are the starred receivers for a given sender. Note that we are aware that the starred process can be improved by precalculating the starred contacts and storing it in a separate data structure (cache). However, as this is a design decision, not related to domain analysis, we will refine our decision later. R28 may seem to be assigned to 'know' concept PendingDeliveries. As this concept has the knowledge needed to check this, we think that needs to check this condition and discard further book requests.

Note that in order to accomplish R12 through R16 we can use the concepts of simple relational database, which is a design decision. However, in order to implement R15 it may be more efficient to use a data

¹We considered R14 as FloorMap (and not BuildingMap) because in this project we are only considering one-floor delivery.

structure such as a priority queue updated with dynamically-adaptable priority. This last feature will allow the system to guarantee a minimum Quality of Service because old requests can become priority. We can use a simple Round-Robin algorithm in order to implement the adaptable priority. At design time we will need to specify these issues.

‘Do’ concepts

Now that we defined boundary and know-like internal concepts, lets focus on do-like concepts.

Before focusing on the problem, however, it is important to note the architecture of our system. It is composed by a central node and a robot, both communicated by a wireless network. Due to this composition, and user stories such as ST-10² force the system to have redundant concepts.

Having the latter architecture in mind, we will discover responsibilities by re-reading the fully-dressed description of use cases. Specially, the great majority of responsibilities can be derived from the BookDelivery, PickupPackage and DeliveryPackage:

- R19: Maintain the status of the user session (implementing too the ADS-BP04 business policy).
- R20: Show the help page to the user when requested.
- R23: When a user searches a receiver (by name, room...), locate all matches.
- R24: Check the password introduced in the robot.
- R25: Check the password introduced when logging in.
- R26: Manage users and user information.
- R29: When managing users, distinguish between the role of administrator and the role of user.
- R30: Make the robot follow the track.
- R31: Make the robot stop at the correct point.
- R32: Start buzzer to notify the user that the robot is waiting for him.
- R33: Maintain a regular contact (from the central server) to the robot, in order to speed up robot failure detection.
- R34: Open the section that is required (in our approach, turn on the LED associated with the section we want to open).
- R35: When the battery level is low, prioritize a movement to go to charging station.
- R36: Determine when to pick up and delivery each package. Needs to be able to recalculate it on-fly, because a delivery may be booked meanwhile the robot is already moving. Note that this responsibility includes the implementation of ADS-BP05 business policy.
- R38: When picking up or delivering packages, determine when the user can be considered idle, and the robot can continue with other deliveries.
- R39: Determine when an initially failed pickup or delivery must be re-tried.
- R40: Determine when a package must be returned to the sender.
- R41: Determine when a booking must be discarded (e.g. after 3 attempts to pick up the package).
- R42: Detect a robot failure
- R43: Allow ‘commands’ to go from the central server to the robot, and allow responses to them to fo from the robot to the central server.
- R44: Coordinate the work and delegate the work requested by users.
- R45: Distribute the work requested by the central server on the robot.

Now, we will try to identify the concepts that can handle these responsibilities.

As suggested in the book, we will have a controller concept that will be responsible of delegating/scheduling the work requested by the actors to other concepts. However, as the architecture specifies two working units, we will have one controller for each unit: RobotController and ServerController. The

²As the administrator, I need to be able to be notified when there are some non-automatically solvable problems. I don’t want to care about other problems.

main controller will be the ServerController, which will be in charge of coordinating all the work and delegating sub-jobs to the RobotController (R44). The latter will manage lower level responsibilities, more tied to the robot domain (R45, R31, R32, R34).

The summary of 'Do' concepts discovered is:

- ServerController (R44)
- SessionController(R19, R20, R23)
- DeliveryCoordinator (R35, R36, R38, R39, R40, R41)
- PasswordChecker (R24, R25)
- ServerCommunicator (R33, R42, R43)
- UserManager (R26, R29)
- RobotController (R45, R31, R32, R34)
- RobotCommunicator (R33, R43)
- TrackFollower (R30)

R19, R20 and R23 are assigned to a new concept: SessionController, because it is its responsibility to control each user interaction with the system. R24 and R25, however have been assigned to a separate concept (PasswordChecker), because this role is differentiated from the session. As R32 is very simple and only implies accessing at the buzzer when certain condition happens, we will assign it to the concept that is aware of this condition: RobotController. The same reasoning is valid for R34. We assigned the responsibility R33 to two concepts, because this responsibility must be distributed in both the robot and the server. As the task mainly implies communication between both, we assigned it to the communicators: ServerCommunicator and RobotCommunicator. R26 and R29 cannot be assigned to the ServerController because both problems are complex enough to qualify for a separate concept. Note that all responsibilities related to robot movement planification (R35, R36, R38, R39, R40, R41 which point should go now, which should go later...) have been assigned to a single concept. Although this may seem a lot of responsibilities for a single concept, we think that refining this to a smaller concepts is a design decision, so we will do in next iterations of the project. Finally, R30 have been assigned to a standalone concept due to its complexity.

6.1.1.3. Summary of concepts

After the discussion that led us identify the concepts, we summarize all these in figure 6.1.

6.1.2. Attribute definitions

The RobotPosition has four attributes: the last known position (important point, such as Office 123), the current position (in case it is stopped), the position it is going to, and an attribute to indicate whether it is stopped or not . This is useful for the server to maintain a record of the approximate position where the robot could be.

SecureBoxContents stores the identifier of the delivery associated with each section of the secure box. Another option would be to store only a reference to the Sender and to the Receiver, but storing the ID of the communication itself allows deducing the sender and the receiver.

PendingDeliveries contains the information about deliveries that are already pending and DeliveryHistory contains the information about finished deliveries.

FloorMap stores the pair {pointNum, Name} for each important point in the floor. One example could be 1, "Office 623".

The UserList contains the ID, name, password, e-mail address and a reference to his/her office for each user in the system.

UserManager contains an attribute of the current logged users that are managing profiles. ServerController does not have a numPending attribute for each user, because this concept can deduce this information from PendingDeliveries.

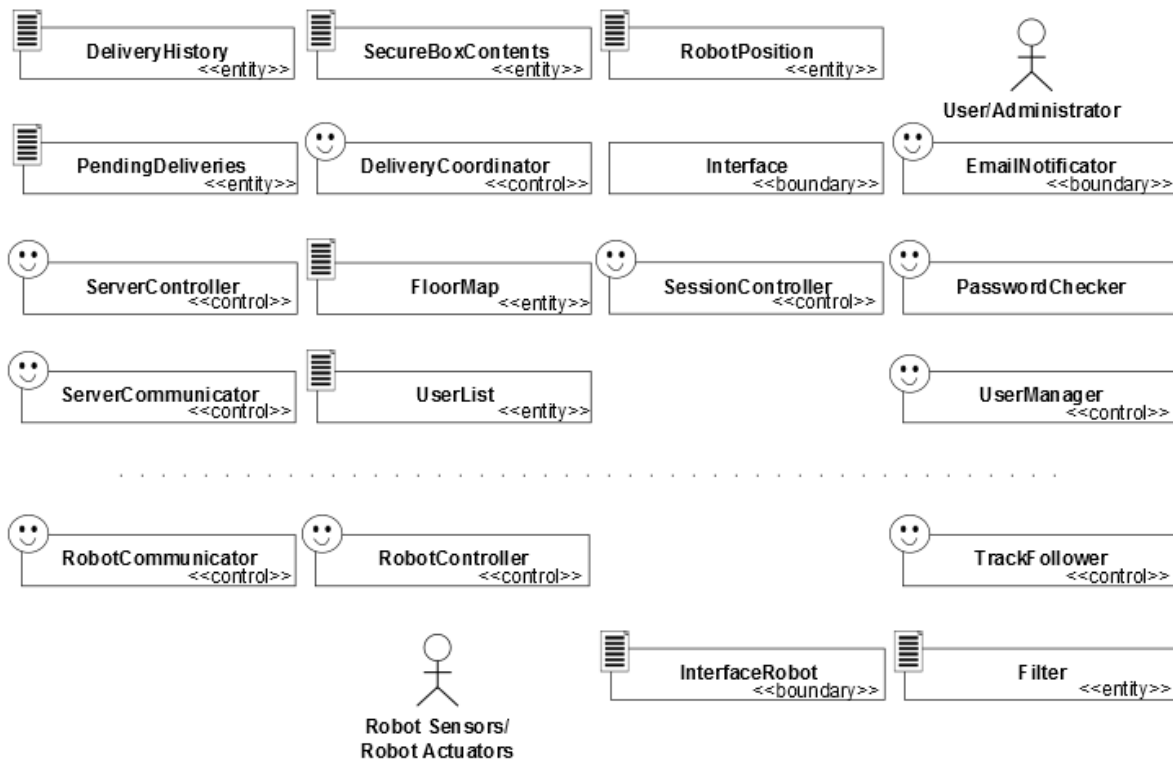


Figure 6.1.: Concept diagram for the Automatic Delivery System.

The DeliveryCoordinator will have one attribute waiting, in order to know its state: it is waiting for a delivery to finish or it can accept and allocate deliveries. It also contains the priority of the current delivery, which is a system parameter derived from the QoS constraint and the battery charging constraint.

ServerController will know the state of the robot too, using isRobotHealthy (to know if server can reach it) and communicating with RobotPosition.

Finally, SessionController contains attributes tied to the user interaction with the system, such as isLoggedIn (to know if the user was logged in correctly), isManaging, isBooking and isTracking (to know the action of the user), isAdmin (to know whether the user is the administrator or not), numDeliveries (a derived attribute from PendingDeliveries to know the number of Deliveries that the user have booked), nonConfirmedBookings (to know the non confirmed bookings). Special mention requires the logging in. As we discussed, we included a password checker to check whether the password introduced by the user is correct or not. On the other hand, we mentioned the maximum number of attempts when talking about the Login use case. Therefore, we need to include two attributes to handle this: numOfLoginAttempts and maxNumOfLoginAttempts. One can think that the correct concept to place these two attributes is PasswordChecker. However, this concept is unique in the system, but there will be several users trying to access to the system. Therefore, we need a concept that will be materialized/instantiated for each user trying to access the system. This concept is the session controller.

Let's move now to the robot's concepts.

RobotCommunicator will maintain the status of the communication in the attribute isServerHealthy.

Filter concept must maintain the current state for each sensor. Also, in order to filter the inputs it can maintain an accumulated state and update it every time it checks the actual input. The currentState attribute will only be updated when the same value have been repeated for more than a specified number of times.

The InterfaceRobot will maintain the keys that have been pressed in the numerical keyboard and the current value for outputs.

TrackFollower must maintain the state of the algorithm for tracking and following the line. More details about this algorithm and its states are provided in section 6.3.1.

Finally, RobotController will contain: isMoving will be useful for determining whether the robot is moving or is waiting for instructions; openedSection will store which section has been opened; and current/last/destinationPosition will store where the robot will go, where it has been and where it is, if it is stopped.

6.1.3. Association definitions

Associations of different concepts mentioned in 5.1.1 are described in Tables 6.1 and 6.2. Note that these association and related concepts derive from Use cases UC-1 to UC-19. Instead of illustrating concepts and their associations for each use cases, we use a full table to describe all related concepts and associations in the system. This is because we believe that we separate the whole system into 19 use cases (it's a lot!), as a result, some of them are so easy to understand, while some of them have very strong coupling that it's hard to separate one from another.

The system has two main kinds of inputs. One is from humans, including registered users, administrators and non-registered visitors. The other is from non-human objects, including sensors (motor sensors, light sensors, magnet sensors) and keypad inputs. At the same time, interactions between human/non-human inputs and outputs are web page or mobile application, which is called Interface, and another kind of interface, which is called InterfaceRobot, to indicate robot-related interactions with human. These kind of interactions may include generating 'buzz' sound and opening designated section of secure box.

From the system point of view, it is mainly divided into two parts, server part and robot part. Interfaces to interact between these two parts are ServerCommunicator in the server side, and RobotCommunicator in the robot side. The communication between ServerCommunicator and RobotCommunicator is mainly to send/receive signals containing operating messages from each side and guarantee QoS of wireless communication.

In the server part, it is composed of Interface, EmailNotificator, SessionController, PasswordChecker, UserManager, RobotPosition, SecureBoxContent, DeliveryHistory, PendingDeliveries, DeliveryCoordinator, ServerController, FloorMap, UserList and ServerCommunicator. Among all these concepts and their associations, ServerController, SessonController, and DeliveryCoordinator play an important role in handling with requests, determining next operations and general management. For the SessionController, it is the core concept regarding web operation, whenever a user wants to add an account, modify his/her user information, or book a delivery, the SessionController will gather request from the user and dispatch them into other modules. For example, when a user request booking a delivery, SessionController will add the delivery ID into PendingDelivery queues. For the association of DeliveryCoordinator with other concepts, it is the decision maker of which delivery needs to be done next, and what the robot needs to do in the next step. Take the initialization of pickup package as an example, when waiting status become false, that is, when the robot is ready for another delivery, the DeliveryCoordinator will assign the first delivery ID in the PendingDelivery queue for as the next delivery task, then, it will access RobotPosition to obtain current location of robot, access FloorMap to determine how the robot will get from current location to the sender's location, send instructions to the Robot through ServerController about the 'move' instructions. For the server controller, it is in charge of sending instructions to robot, creates new delivery if necessary (e.g. when one delivery trial fails and needed to be delivered again), updating RobotPosition and DeliveryHistory and notify EmailNotificator to send email to user about a email about to come.

From the robot part, it is composed of RobotCommunicator, RobotController, TrackFollower, Filter and InterfaceRobot. RobotCommunicator transfer message to RobotController, then RobotController will interact with the TrackFollower, if the message is about making robot move; the RobotController will interact with InterfaceRobot if the message is about to turn on a LED, or trigger the buzz. Filter will sample and obtain sensor states. TrackFollower will request state information from the Filter about sensors to move to the designated position or stop immediately.

The final domain model diagram is shown in figure 6.2.

Concept Pair	Association Description	Assoc. name
SessionController ⇔ Interface	SessionController accesses information from the Interface about user information and possibly updates these information	Accesses, updates
SessionController ⇔ PasswordCheck	SessionController activates and uses PasswordChecker in order to verify user's identification in the case user needs to log in	Activates, uses
SessionController ⇔ PendingDeliver:	SessonController updates and stores booking information to PendingDeliveries in the case a user books a delivery	Updates, stores
ServerController ⇔ SessionController	SessionController notifies new booking information to the ServerController. ServerController may also creates new booking delivery to the SessionController in the case the pickup or delivery process failed at this time and need to re-do the process, or delivery failed for three times and a package need to be returned to the sender	Notifies new booking, creates
ServerController ⇔ Password-Checker	ServerController activates and uses PasswordChecker in the case a sender or a receiver enters his/her password for authentication	Activates, uses
PasswordChecker ⇔ UserList	Password checker requests password from UserList to check whether the password that the user inputs in the robot's keyboard is consistent with expected password	Requests password
SessionController ⇔ UserManager	SessionController activates and uses UserManger to provide detailed user management function (such as add or remove user account)	Activates
UserManager ⇔ UserList	UserManager accesses and creates UserList when a user choose to add an account. UserManager accesses and modifies UserList to change user's information	Accesses, creates, modifies
SessionController ⇔ DeliveryHistory	SessionController accesses DeliveryHistory to get information about delivery history, in the case user want to log in and check delivery information	Accesses
DeliveryCoordina ⇔ RobotPosition	DeliveryCoordinator accesses RobotPosition to evaluate current position of robot and determine action of next step for the robot (move, stop or enter error state)	Accesses
ServerController ⇔ RobotPosition	ServerController updates position information to RobotPosition, in order to store current detailed position status	Updates
DeliveryCoordina ⇔ SecureBox-Content	DeliveryCoordinator accesses and updates SecureboxContents in the case the sender just put a package into an available section of secure box or a receiver just takes a package out from a section	Accesses, updates
ServerController ⇔ DeliveryHistory	ServerController updates DeliveryHistory by transmitting delivery checkpoint (such as finish pickup, begin delivering, package delivered) to the DeliveryHistory	Updates
DeliveryCoordina ⇔ DeliveryHistory	DeliveryCoordinator updates delivery history information to DeliveryHistory for refreshment	Updates

Table 6.1.: Association Definitions (part 1)

Concept Pair	Association Description	Assoc. name
ServerController ⇔ DeliveryCoordinator	ServerController starts DeliveryCoordinator at the beginning of running a queue of delivery task; DeliveryCoordinator notifies ServerController about the robot's action, i.e. what should robot do next, to move or to stop.	Starts, notifies robot action
PendingDeliveries ⇔ DeliveryCoordinator	PendingDeliveries pop up the first delivery information in the queue to the DeliveryCoordinator in the case 'waiting status' is false, i.e. the robot is idle and ready for next task	Retrieves, updates, deletes
DeliveryCoordinator ⇔ FloorMap	DeliveryCoordinator accesses FloorMap to get the correspondent office location according to the designated sender or receiver	Accesses
DeliveryCoordinator ⇔ FloorMap	UserList request information from FloorMap to get name for a PointNum	Request Name for a PointNum
ServerController ⇔ ServerCommunicator	ServerController contact with ServerCommunicator to sending and receiving message	Sends and receivers
ServerCommunicator ⇔ RobotCommunicator	ServerCommunicator sends and receives signals to RobotCommunicator so that Server side and robot side can interact with each other	Sends and receivers
TrackFollower ⇔ Filter	TrackFollower requests values from Filter so that it can get sensor states (especially those sensors regarding motor and light)	Requests values
RobotController ⇔ TrackFollower	RobotController controls the start and stop of the robot by sending start/stop instruction to trackfollower	Starts and stops
RobotCommunicator ⇔ RobotController	RobotCommunicator notifies messages from server side and requests sending messages to server (as an intermediary)	Notify messages from server and request sending messages to server
RobotController ⇔ InterfaceRobot	RobotController updates outputvalues to InterfaceRobot in the case LEDs need to be turned on of turn off (as an indicator that section is open or close)	Update output values
RobotController ⇔ Filter	RobotController requests sensor values(especially magnetic sensor value) in order to detect current location	Requests values
Filter ⇔ InterfaceRobot	Filter requests values from InterfaceRobot to request information about state of each input sensor	Requests values

Table 6.2.: Association Definitions (cont.)

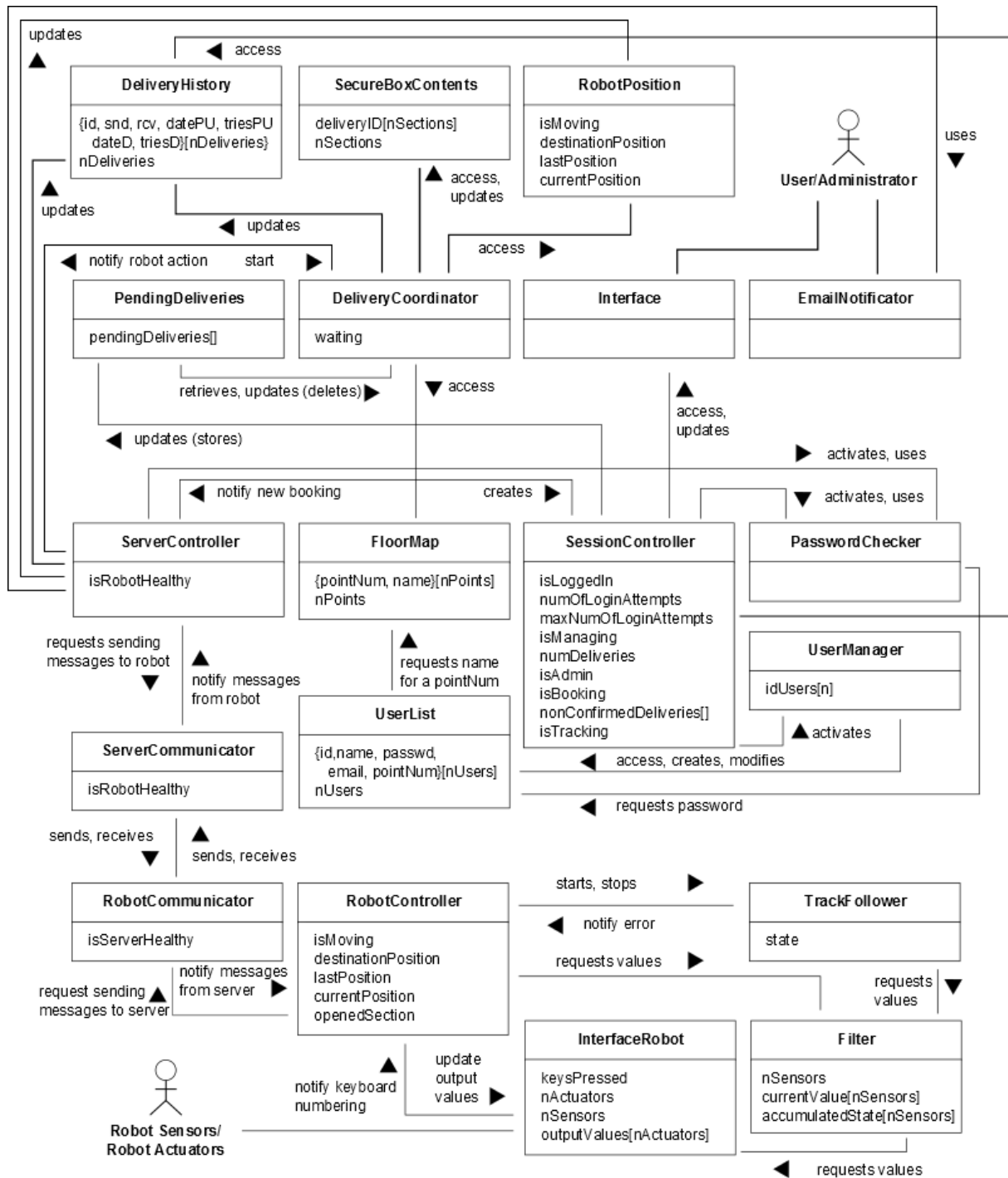


Figure 6.2.: Domain model diagram for the Automatic Delivery System.

UC	PW	DeliveryHistory	SecureBoxContents	RobotPosition	PendingDeliveries	DeliveryCoordinator	Interface	EmailNotificator	ServerController	FloorMap	SessionController	PasswordChecker	UserManager	UserList	ServerCommunicator	RobotCommunicator	RobotController	InterfaceRobot	TrackFollower	Filter
UC1	9	x			x		x		x		x	x		x						
UC2	3	x			x		x		x		x	x		x						
UC3	1						x													
UC4	1						x						x	x						
UC5	3	x			x		x		x		x	x		x						
UC6	1						x				x	x		x						
UC7	5	x	x	x	x	x			x	x					x	x	x	x	x	x
UC8	5	x	x	x	x	x		x	x	x					x	x	x	x	x	x
UC9	1						x						x	x						
UC10	1						x						x	x						
UC11	6								x			x		x	x	x	x	x		
UC12	3							x												
UC13	2							x												
UC14	1					x			x						x	x	x	x		
UC15	10			x					x	x					x	x	x		x	x
UC16	10			x					x	x					x	x	x	x		
UC17	10			x					x	x					x	x	x	x		
UC18	10																x	x		x
UC19	1						x						x	x						
Max PW		9	5	10	9	5	9	5	10	10	9	9	1	9	10	10	10	10	10	10
Total PW		25	10	40	25	11	21	10	62	40	16	22	4	26	47	47	57	47	20	30

Table 6.3.: Traceability matrix

6.1.4. Traceability matrix

The traceability matrix for is shown in Table 6.3.

6.2. System Operation Contracts

Table 6.9 shows the system operation contracts for Track Delivery Status. Given that the actor fulfilled the preconditions for this operation, the operation postconditions specify the guarantees of what the system will do. The precondition for a user is that he or she should log in first, request tracking status; and the ‘interested delivery’ information, such as sender, receiver, and date is stored in the system database. Given these preconditions, user can check the interested delivery information in the web user interface (UI). After that, tracking request in Session Controller is reset to false.

Table 6.10 shows the contracts for the operation Authentication to Robot. The system should be fair, so each user should be allowed the same number of attempts (MaxNumofAttempts). Thus, the precondition for a new user is that NumofAttempts equals to zero. Besides, if NumofAttempts does not exceed the MaxNumofAttempts, the precondition of authentication operation is satisfied. The postcondition for the system is that, after the current user ends the interaction with the robot, the NumofAttempts is reset to zero and current instance of the password object is destroyed.

Table 6.11 shows the contracts for the operation Notify Receiver. If actor fulfilled the preconditions for this operation, the operation postconditions will specify what the system will do. The precondition in

Name:	BookDelivery
Responsibilities	Enables the user to book a delivery system.
Cross References	UC-1
Preconditions	a) An authenticated user logged in to the system. b)The user made a delivery request and the delivery machine arrived to a place where the user can book his/her delivery without moving from its position. c) The user is currently in his/her place to book his/her delivery.
Postconditions	The delivery request is completed and the delivery task of the package booked by the user is issued to the delivery machine task queue.

Table 6.4.: System Operation Contracts for BookDelivery

Name:	MoveRobotToPoint
Responsibilities	Moving the robot to a desired location.
Cross References	UC-15
Preconditions	a) A desired destination point of the delivery machine is decided by the system. b)Desired location is within the working field of the delivery machine. c) System is able to find the current position of the delivery machine without any exception. d) Move to destination task is at the top of the delivery machine task queue. e) Nothing is preventing the mobility of the delivery machine.
Postconditions	Delivery machine moves to the desired location point and ready to fetch the next task from its task queue.

Table 6.5.: System Operation Contracts for MoveRobotToPoint

Name:	SetMotorRSpeed & SetMotorLSpeed
Responsibilities	Sets the rotation speed of the motor 1 which is in the right side of the robot.
Cross References	UC-16 UC-17
Preconditions	a) The system decides to change the speed of motor 1—2. b)Desired speed of Motor 1—2 is within the acceptable limits which is governed by the specifications of the motors going to be used. c) Motor 1—2 is working properly. d) Speed of the Motor 1—2 can be effectively configured by a corresponding driver.
Postconditions	Speed of the Motor 1—2 is changed to the desired level.

Table 6.6.: System Operation Contracts for SetMotorRSpeed & SetMotorLSpeed

Name:	PositionInspection
Responsibilities	Checks the position of the delivery machine and corrects its position if necessary.
Cross References	UC-18
Preconditions	he reflective sensors and magnetic sensors mounted on the delivery machine is working properly and their values can be received and processed at a remarkable frequency for the fineness of the position inspection.
Postconditions	Position of the delivery machine is inspected and necessary delivery machine position correction task is issued to delivery machine in a way that the other tasks of the delivery machine is interrupted by this task. In other words, position correction task is inserted at the bottom of the delivery machine task queue.

Table 6.7.: System Operation Contracts for PositionInspection

Name:	ChargeBattery
Responsibilities	Performs the automatic charging feature of the system.
Cross References	UC-14
Preconditions	a) Battery level indicator is working properly and its output showing the charge level of the battery as percentage level can be received by the system. b)System receives a 1st or 2nd (explained in the fully dressed explanation of UC-14)level battery level warning from the battery level indicator. c) There is a previously specified charging place appropriate for the delivery machine. d) There is no electricity cut happening during this operation.
Postconditions	Delivery machine battery is fully charged and ready to continue its task queue.

Table 6.8.: System Operation Contracts for ChargeBattery

Operation:	Track Delivery Status
Preconditions	<ul style="list-style-type: none"> ● IsLoggedIn in Session Controller is true. ● IsTracking in Session Controlle is true. ● The “interested delivery” nformation (ID, Sender, Receiver Date) is stored in the system database (Delivery History or Pending Delivery).
Postconditions	<ul style="list-style-type: none"> ● IsTracking set false in Session Controller ● Specific delivery status is displayed in the User Interface.

Table 6.9.: System Operation Contracts for Track Delivery Status

Operation:	Authentication to Robot
Preconditions	<ul style="list-style-type: none"> • Set of valid passwords stored in the system database is non-empty. • NumofAttempts = 0, for the first attempt of current user. • NumofAttempts \leq MaxNumofAttempts
Postconditions	<ul style="list-style-type: none"> • NumofAttempts = 0, if the entered password \in Valid keys. • Current instance of the password object is archived and destroyed.

Table 6.10.: System Operation for Authentication to Robot

Operation:	Notify Receiver
Preconditions	• IsBooking in the Session Controller is true.
Postconditions	Receiver receives the notifying message.

Table 6.11.: System Operation for Notify Receiver

this operation is that the sender firstly have finished booking delivery, IsBooking in the Session Controller is true. Given this preconditions, system will automatically send a notifying message to receiver saying that his or her package is on the way.

Table 6.12 indicates the system operation contract for ObtainHelp. For the function of providing help instructions to user, the implementation is quite easy and clear. There is no precondition for the system if the user is visiting the user interface and is willing to obtain help. After the user click 'Help' link in the web interface, it will turn to help page, which contains instructions about how to use the Automatic Delivery System.

Table 6.13 indicates the system operation contract for PickupPackage. For the function of pickup package for the sender, certain preconditions and postconditions must be guaranteed. The system should wait for the 'waiting status'(which is an attribute of concept 'DeliveryCoordinator') to become 'False', which means the robot is now ready for new pickup and deliveries, before doing the 'Pick up Package' operations. Another precondition is there exists pending deliveries in the queue, which means there are 'Pickup' work waiting to be done. Also, Pickup package need one section from the secure box to contain package, so at least one section should be available. Other preconditions include both server and robot should be able to communicate with each other. After picking up the package, the system enters the process of delivering the package which just picked up from the sender, so any other delivery service should be pending, so 'waiting status' should become true. Also, this delivery is no longer in line waiting for picking up, so the delivery ID should be canceled from the pending delivery queue. This new delivery ID should be linked with the section number that contains the package. We should always guarantee communication between server and robot available if certain operations is related to robots. So IsRobotHealthy and IsServerHealthy should always be true under this circumstance.

Table 6.14 indicates the system operation Delivery(for success scenario). For the function of Delivery package to receiver, certain preconditions and postconditions needed to be accomplished. Before Delivery is executed, We should guarantee the waiting status is still true, in this way, no other instructions will

Operation:	ObtainHelp
Preconditions	None
Postconditions	User interface switches to help page.

Table 6.12.: System Operation Contracts for ObtainHelp

Operation:	PickupPackage
Preconditions	<ul style="list-style-type: none"> • Waiting status = False; • Pending Deliveries set $\neq \emptyset$; • Available Sections set of SecureBox $\neq \emptyset$; • IsRobotHealthy = True; • IsServerHealthy = True.
Postconditions	<ul style="list-style-type: none"> • Waiting status = true; • Last DeliveryID is excluded from Pending Deliveries set; • New ID linked with section number x is added in the Secure-BoxContents set; • State of confirmbutton = invalid; • IsRobotHealthy = True; • IsServerHealthy = True;

Table 6.13.: System Operation Contracts for PickupPackage

Operation:	Delivery (for success scenario)
Preconditions	<ul style="list-style-type: none"> • Waiting status = True; • IsRobotHealthy = True; • IsServerHealthy = True; • IsRobotHealthy = True; • Destination Position = receiver's location.
Postconditions	<ul style="list-style-type: none"> • Waiting status = False; • The link between finished delivery ID and it's package section is excluded from the SecureBoxContents set; • Update Delivery History, delivery of certain ID should be marked as 'finished'; • state of confirmbutton = invalid; • IsRobotHealthy = True; • IsServerHealthy = True.

Table 6.14.: System Operation Delivery (for success scenario)

Operation:	UserManage
Responsibilities	Display the user or administrator with the option of add user, remove user and edit user information
Preconditions	The user or administrator gets authorization from system IsManaging=0; IsLoggedIn=0 IsAdmin=0 —for Administrator IsAdmin=0 —for users
Postconditions	The user chooses one of options from the system interface IsManaging=1; IsLoggedIn=1 IsAdmin=1 —for Administrator IsAdmin=0 —for users

Table 6.15.: System Operation UserManage

Operation:	AddUser
Responsibilities	To add a new employee information to the system database
Preconditions	The employee information is not in the system IsManaging=1; IsLoggedIn=1 IsAdmin=1 —for Administrator IsAdmin=0 —for users
Postconditions	The system database records the new employee information IsManaging=1; IsLoggedIn=1 IsAdmin=1 —for Administrator IsAdmin=0 —for users

Table 6.16.: System Operation AddUser

be called to make the robot move to other place for a package pickup. Also the system should set destination position to the receiver's location (this is obtained from the FloorMap with point number and its corresponding point name). Also, communication between server and robot should always be ensured. After finishing Delivery operation, the waiting flag should be marked as 'False', so subsequent delivery task will be assigned and robot will move on for the future work. Also, since the section that contain last receiver's package has been emptied, so the status of this section should be marked as 'available' now for future delivery use. The system should also update delivery history for this entire delivery process is finished. At the same time, we should always guarantee the communication between robot and server is normal, since all previous operations is dependent on the communication between these two objects.

This UserManage contracts express the attribute of SessionController in the main domain model. Every users are equally provides with login window when he or she want to get access to the UserManage interface. Sytem will then automatically authenticate whether the user is administrator or not. The precondition is that the IsManaging and Isloggedin are zero at the start points. After success login, the IsManaging and Isloggedin are equal to 1, and IsAdmin equal to 1 for administrator and equal to 0 for users.

In the preconditions for the operation add user (table 6.16), the status value of IsManaging and Isloggedin is 1 for user or administrator. Differences value of IsAdmin exists between user and administrator. in the postconditions, the status value remains the same for users has not logged out and still has the authorization to manage the user information

Operation:	RemoveUser
Responsibilities	To remove an existing employee information to the system database
Preconditions	The employee information is in the system IsManging=1; IsLoggedIn=1 IsAdmin=1 —for Administrator IsAdmin=0 —for users
Postconditions	The system database remove the previous empolyee information IsManaging=1; Is LoggedIn=1 IsAdmin=1 —for Administrator IsAdmin=0 —for users

Table 6.17.: System Operation RemoverUser

Operation:	Login
Preconditions	The valid names/keys stored in the system database is non-empty; OperatingInterfaceStatus =Normal; The user has already created a user account in the system. numOfAttempts < maxNumOfAttempts numOfAttempts = 0, for the first attempt of the current user
Postconditions	numOfAttempts = 0, if the entered user name/password Î valid user name/password current instance of the user name/password object is archived and destroyed LoginStatus=Success

Table 6.18.: System Operation Login

Operation:	AccessDeliveryRecord
Preconditions	The delivery record stored in the system database is non-empty; OperatingInterfaceStatus=Normal; LoginStatus=Success.
Postconditions	Delivery records can be displayed chronologically in the displayer; Administrator can edit the delivery records in the system.

Table 6.19.: System Operation Access to delivery record

Operation:	NotifyError
Preconditions	SystemStatus=problem; LoginStatus=Success, for user;
Postconditions	Administrator receives notification email about the problem; User finds the notification after logging into the system.

Table 6.20.: System Operation NotifyError

6.3. Mathematical Model

Basically, our system needs two mathematical models, described in the following sections.

6.3.1. Track detection

In this part, the question, “How will the robot will follow the path ?”, is going to be answered.

Our system will assume the track formed basically by two lines on the road will be provided. This track is going to be simple plain white line for the delivery machine to follow. White line will be recognized and followed by the help of two optical sensors . We need two optical sensors to achieve turning the machine and also to keep the machine follow the line in a smooth manner (figure 6.7) rather than making zig zag (figure 6.8). We mean by smooth manner that the machine moves almost in a straight line and makes small oscillations around the moving axis so that these oscillations are almost negligible for the movement and the scene. There are simply 2 things which need to be considered while detecting the track and maintaining a successful path follow;

1. The machine movement should be step oriented. To explain more, system should firstly detect its position related with the track and then direct the machine to do a corresponding movement. This can be understood by considering two simple scenarios.
 - a) By getting '1' (representing the sensor is sensing the white line) from both optical sensors, system understands the road is flat and the machine needs to go forward (figure 6.4).
 - b) By getting '1' from one of the sensors and '0' from the other this means the machine is going away from the path which is bad. So correspondingly system directs the machine to move again into path (figure 6.5).
2. If the machine gets out of track (figure 6.6) and cannot sense any line with its optical sensors, system should be warned by the machine and the machine should go into 'wait' state. In this state, machine suspends all of its tasks and waits for the administrator to fix the problem by removal of the machine on the track.

The general moving states of the delivery machine is as the following (figure 6.3).

6.3.2. Path resolution

In this part, the strategy of the system for pick up and hand in packages operations of each delivery task will be presented.

A path (figure 6.9) is simply the road which has a starting and an ending point. In our case starting point will be the current position of the delivery machine once the delivery task is fetched. The ending point will be the specific places arranged in the users' offices for pick up and hand in operations. This specific place can be simply anywhere appropriate for the user to be able to receive or submit his/her packages without moving his/her desk.

The problem of path resolution for the delivery machine can be simply conceived as a shortest simple path problem. Of course to utilize from this previously solved shortest path problem, our system should be able to first extract the movement map as a simple graph. Vertices of the graph in this case are possible destinations of the machine and the edges are the roads between these possible destinations. Also we need to define a positive "weight" for each edge of the graph for the sake of representing distances between the destinations.

System is going to create the graph by the input of the vertices, connection and edge weights entered by the administrator. Automatic map (graph) extraction is omitted because it is redundant in the sense that the working environment of delivery machine will not be dynamic and large. The destinations in actual road are represented by the 'magnets' which are put on the machine track in a logical way.

After the graph formation of the working environment, path resolution turns into a simple shortest path problem. There are many algorithms for solving this problem (reference [48]) but we will use Dijkstra's algorithm. Even though there are some other more efficient algorithms used but we chose Dijkstra's since

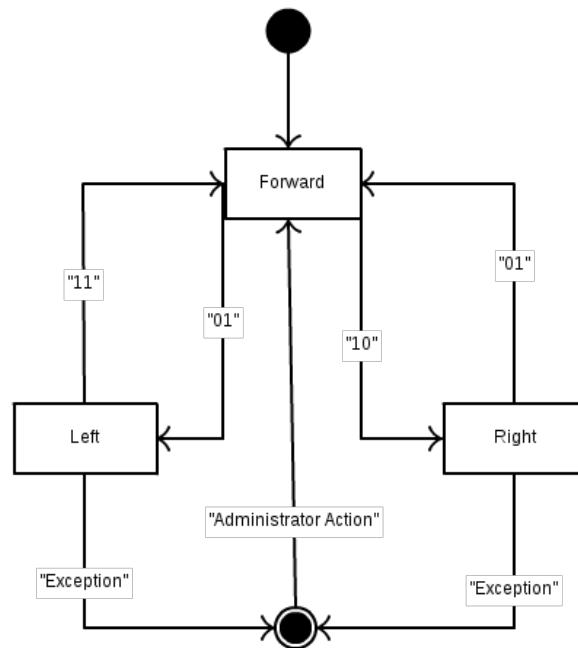


Figure 6.3.: Finite state machine showing the moving cases of the delivery machine and their interactions.

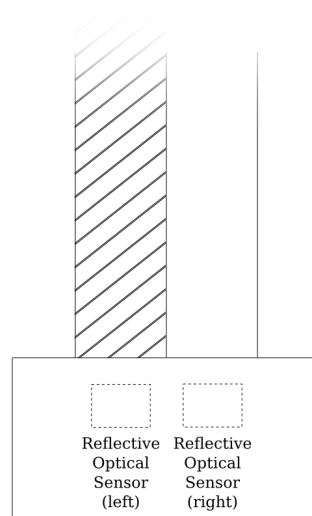


Figure 6.4.: Representation of the machine going forward case.

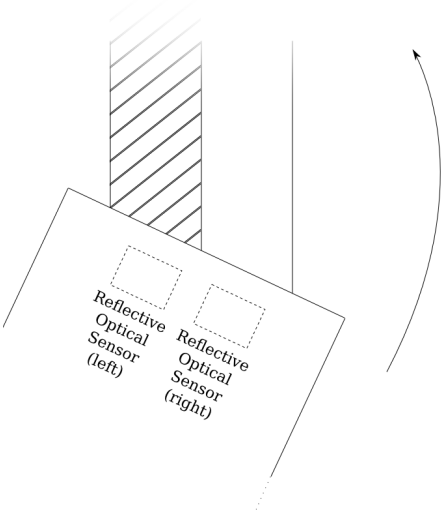


Figure 6.5.: Representation of the machine turning case.

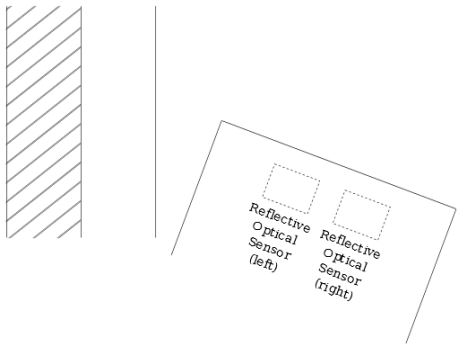


Figure 6.6.: Representation of the machine going out of path case.

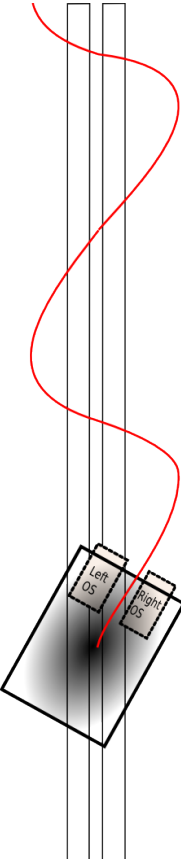


Figure 6.7.: Representation of the machine making unstable zig zags while moving.

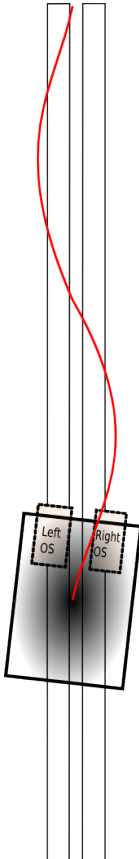


Figure 6.8.: Representation of the machine making small zig zags while moving.

Notation	Explanation
$c(x,y)$	Link cost from node x to y ; $= \infty$ if not direct neighbors
$D(v)$	Current value of cost of path from source to dest. v
$p(v)$	Predecessor node along path from source to v
N'	Set of nodes whose least cost path definitively known

Table 6.21.: System Operation Contracts for BookDelivery

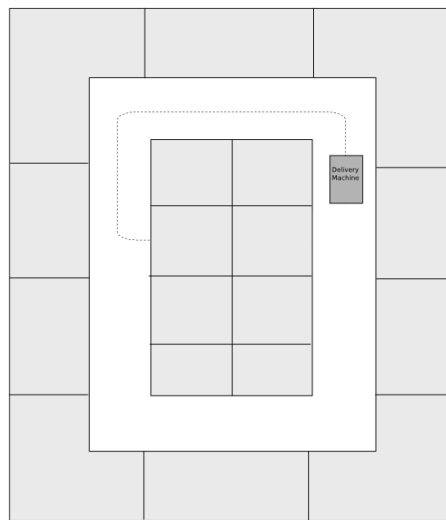


Figure 6.9.: Representation of a simple path of the machine 1.

it is simple to implement, fast and the maps of the working environment for our system is not so complex at all. To briefly explain Dijkstra's shortest path solution algorithm;

Dijkstra's Algorithm is a Link-State Routing Algorithm. It computes the least cost paths from one node ('source') to all other nodes which means basically it gives forwarding table for that node. Actually since we want to find the forwarding path to just one vertex, we will slightly modify this algorithm for it to find the path to just one destination for each time a path resolution requested. Therefore, it will be efficient performance wise since path resolution to one vertice requires just one iteration.

Before presenting Dijkstra's algorithm, some notation should be introduced first.

Depiction of the Dijkstra's Algorithm in pseudo code can be found in the followings (figure 6.10) (reference [54]). Also an example to show how Dijkstra's Algorithm works to find the shortest path between 'u' (source) and 'z' (destination) can be found in the followings (figure 6.11) (reference [54]).

Dijkstra's Algorithm

```

1 Initialization:
2 N' = {u}
3 for all nodes v
4   if v adjacent to u
5     then D(v) = c(u,v)
6   else D(v) = ∞
7
8 Loop
9 find w not in N' such that D(w) is a minimum
10 add w to N'
11 update D(v) for all v adjacent to w and not in N' :
12   D(v) = min( D(v), D(w) + c(w,v) )
13   /* new cost to v is either old cost to v or known
14     shortest path cost to w plus cost from w to v */
15 until all nodes in N'
    
```

Figure 6.10.: Dijkstra algorithm.

Dijkstra's algorithm: example

Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x	2,x	∞	∞
2	uxy	2,u	3,y	4,y	4,y	∞
3	uxyv	2,u	3,y	4,y	4,y	∞
4	uxyvw	2,u	3,y	4,y	4,y	∞
5	uxyvwz	2,u	3,y	4,y	4,y	∞

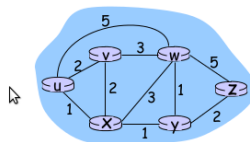


Figure 6.11.: Dijkstra example.

7. INTERACTION DIAGRAMS

We will describe the most important use cases: BookDelivery, PickUp and Delivery.

7.1. Login and BookDelivery

The interaction diagram for BookDelivery use case is simply showing the interaction between the objects for logging in stage of the user and delivery booking by the user.

Firstly for the logging in stage, the interactions happening are as follows;

- Once the user launches the program it will be welcomed by a simple 'login' screen. In this screen user will be asked to enter his/her unique username and password for authentication. After the username and the password is entered in the related parts and the login is initiated, ServerController will be urged to create the SessionController which is responsible for managing this login process.
- Once the SessionController is created, it is going to create the PasswordChecker object to use it as a inquiry to the database of UserList. SessionController uses the checkPassword(username, password) function of the PasswordChecker to find out whether there is a user with the entered username and if there is whether the password is correct.
- After the result of this user authentication done by the PasswordChecker, SessionController interacts with the user interface to show the corresponding messages ("Login success message" or "Login failed message") to the user.

After the user is successfully logged in, the next stage is the delivery booking. The interactions happening for this part are as follows;

- Once the user makes a search for the target user by writing the his/her name, this request is handled by the SessionController. For handling this request, SessionController inquires the database of UserList and if the query results in success, SessionController tells the Interface to show the name of the target user in the target user list.
- Then user can select which user s/he wants to add it to the target user list. This addition of the selected user to the target list and showing this addition to the user in the interface is also handled by the similar interaction in 4. between the SessionController and the Interface.
- The searching and selecting steps (previous two steps) can be repeated till the user finds all recipients he/she want to send packages to. After finishing adding target users to the target user list, user can start book deliveries by clicking the 'confirm' button in the window. The confirmation of the user is handled again by the SessionController

After the deliveries-to-be booked are confirmed, the interactions happening will be as follows;

- Once the user initiates the confirmation of the book delivery requests related with the target list s/he created, SessionController will handle this confirmation request and call the book delivery size check function of the ServerController. This is simply checking whether the user is trying to book more deliveries than the previously specified delivery limit for each user. This limiting is necessary for the system to be fair serve for every user.

If the ServerController's so-called checking function inspects that the user is trying to go beyond the limit, the confirmed deliveries will not be booked by the system. In that case, a corresponding error message will be shown to the user. Else these confirmed deliveries are booked by the ServerController via adding these deliveries to the database PendingDeliveries. After this successful booking operation, again first the ServerController informs the SessionController and SessionController informs the user about the success.

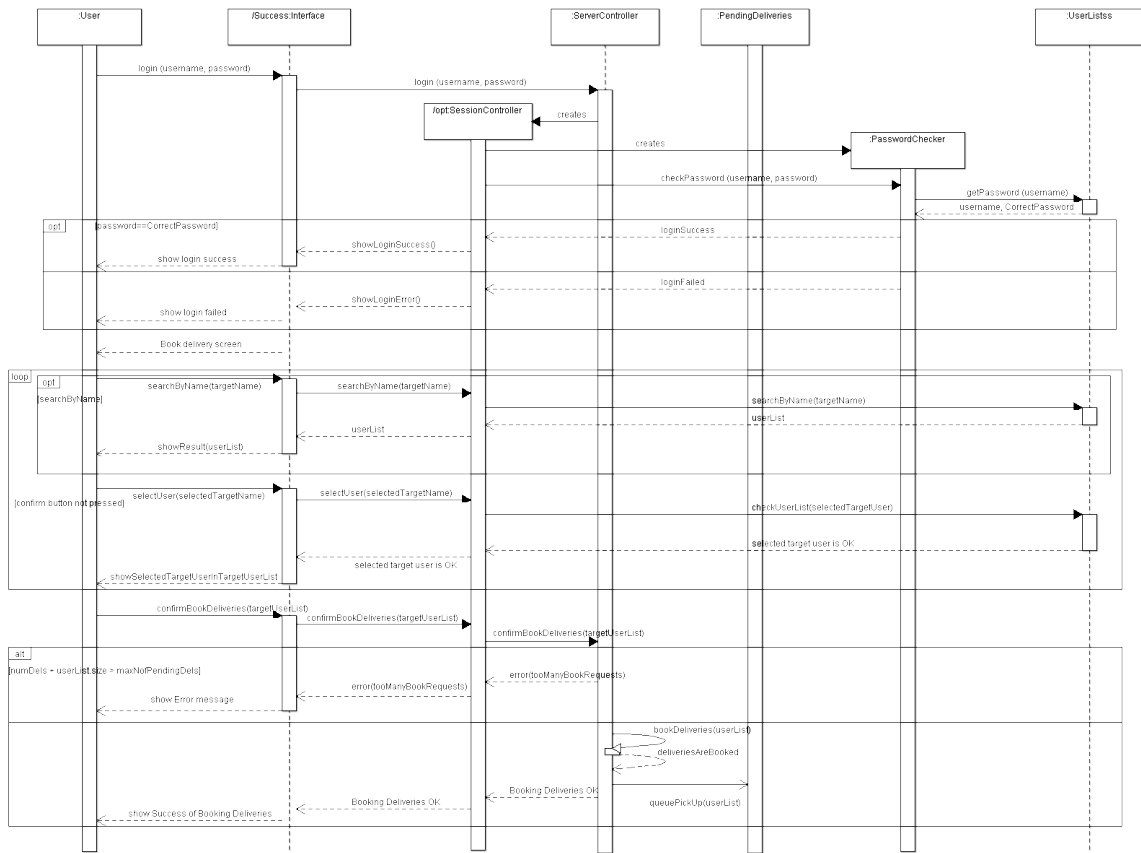


Figure 7.1.: Interaction diagram for the Main part of the book delivery use case scenario.

7.2. PickUpPackage and Delivery

As long as these two use cases have similar functionality and are very tied, we will describe both together. Figure 7.2 shows the main part of the interaction diagram for these two use cases from the perspective of ‘server’ side. Figure 7.3 shows the interaction diagram to obtain the package information with highest priority, which is abbreviated as ‘getMostPriorityDelivery’. This is an important function to complete the process of PickUpPackage or Delivery in the server side. Detailed description of this function can be found later in this section. Figure 7.4 explains how the use case moveRobotToPoint works, which means how the robot can move from one office to the next one and interact with the server. Figure 7.5 shows interaction diagrams of how the server and robot behave when the robot has arrived at the PickUpPackage or Delivery location and wait for picking up the package or giving the package to recipient. Figure 7.6 describes how the robot is capable of following the black line which was set up as track of the robot. In the following paragraphs we will describe the full functionality for these two important use cases – PickUpPackage and Delivery. We will also discuss the decisions that helped achieve this.

First, the serverController creates an instance of the DeliveryCoordinator concept¹, and when this happens the pickUpPackage/Delivery use case begins. Note that if we consider the system as a whole, the ‘initiating’ actor for both use cases is the Timer, because no human factors participate in the initiating part of these use cases. However, if we focus a bit more on the internals of the system, i.e. from the perspective of the system, the Timer is not the only one that initiates the PickUpPackage or Delivery use cases: there are prerequisites that some sequence must happen before a concrete PickUpPackage or Delivery use case starts. A simple scenario could be, for example, that the delivery must be booked and the robot needs to be stopped in the initial point. As we discussed in Chapter 6 and the multiple delivery User Story, this would bring about very bad performance, because the time needed for previous deliveries can be unpredictable and we might have no idea for how long the robot will go back to the initial point. Therefore, we improve the algorithm by allowing the initiation of a PickUpDelivery/Delivery when the robot is in the middle of another delivery process. In other words, we can’t just calculate a “pick up/delivery” plan (e.g. go to point 1, then to point 4, then to point 6 and finally to point 10; finally, return to point 0) and make the robot follow this fixed route. It’s not flexible enough. We want the system to be able to calculate the route dynamically. At a first glance, this can be done basically by two methods:

- Every time a pickup/delivery is booked, recalculate the long-term route.
- Every time the robot reaches a point, look if there are pending deliveries from that point.

The first approach might seem more suitable, because the second one introduces some problems apparently non-easily solvable. However, if we analyze the problems that arise with the first approach we will see that most of them are outside of the control of the system. For example, what if the sender/receiver doesn’t appear in the specified period of time, during the pickup/delivery? Then, the whole long-term route will be affected. These kind of problems would make the routes to be recalculated again and again. On the other hand, the second approach can be easily implemented by defining two simple constraints:

- The robot will always follow a definite path.
- The robot will always move around if there are pending pickup/deliveries. If not, the robot will stay in point 0.

Note that the first was already defined as a global constraint of the system in the first report. Then, we only need to achieve the second goal, which can be implemented with simple controls (in comparison of the controls needed for the first approach). For this reason we decided to choose the first approach. This decision makes the control for the PickUp/Delivery use cases to be initiated only once, when the system starts. This ‘control’ will dynamically initiate the actual concrete use cases when needed.

¹Note that we are not talking about classes here, we are dealing with concepts.

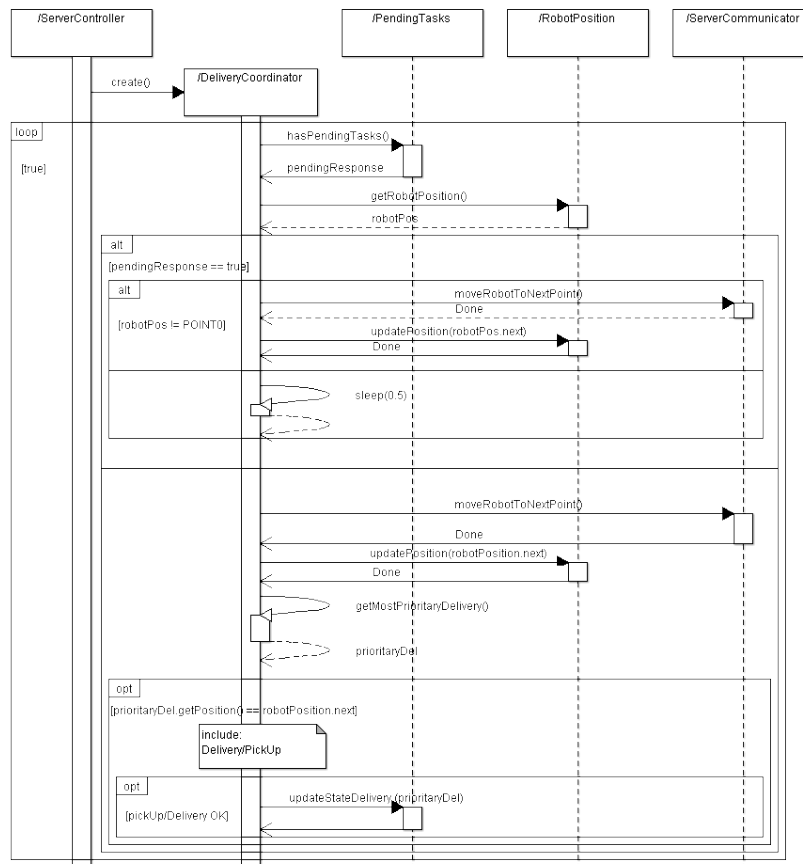


Figure 7.2.: Interaction diagram for the Main part of the pick up and delivery use case scenario.

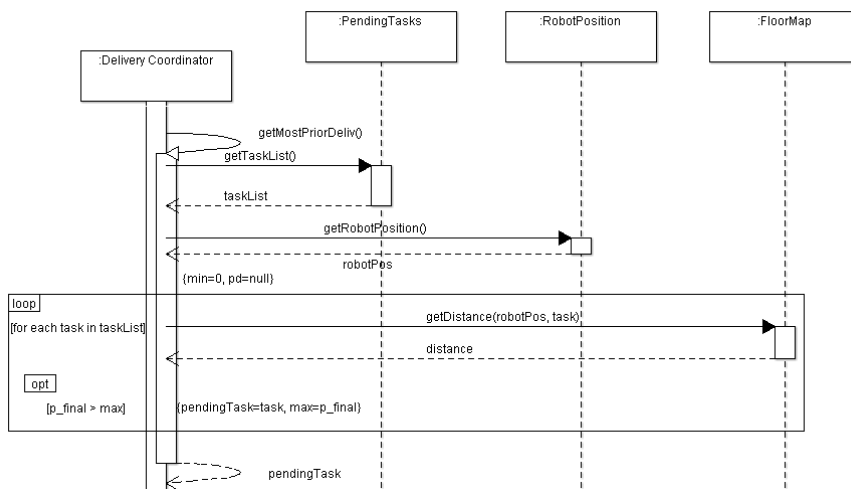


Figure 7.3.: Interaction diagram for the getMostPriorityDelivery function.

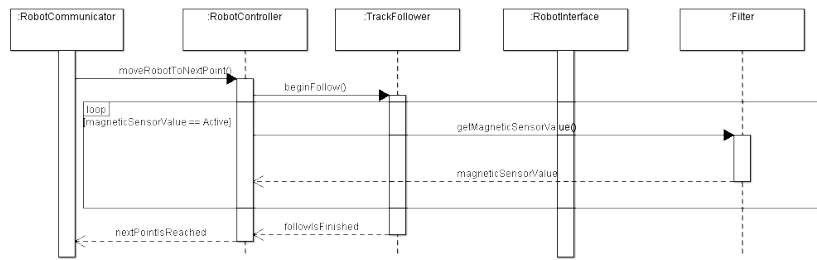


Figure 7.4.: Interaction diagram for the moveRobotToPoint function.

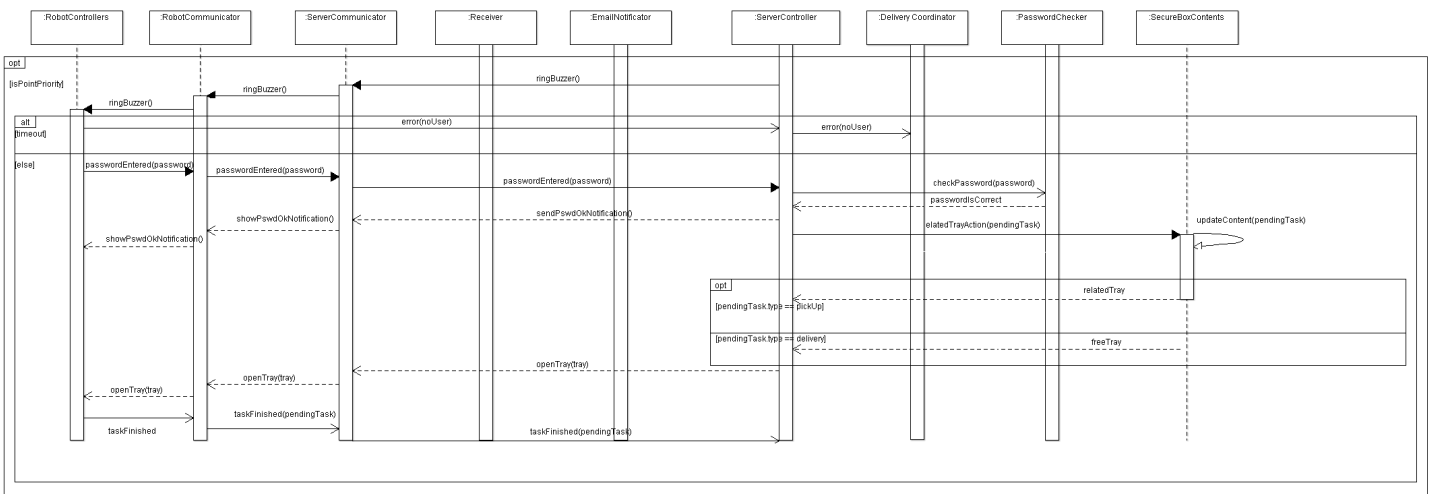


Figure 7.5.: Interaction diagram for the concrete actions that the system does when picking up and delivering an object.

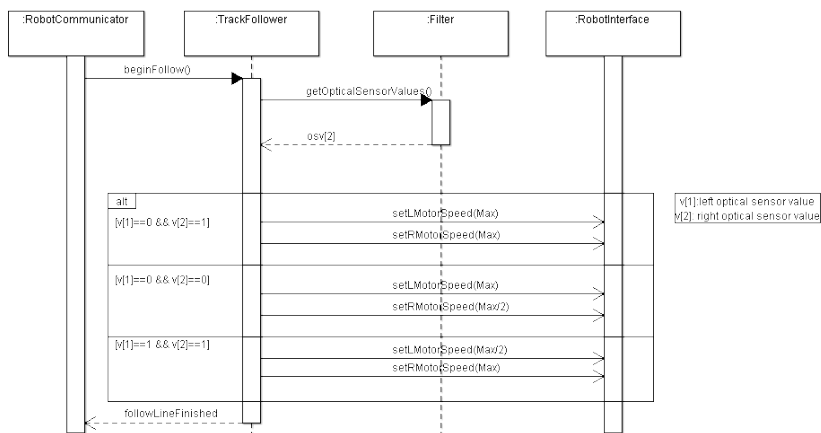


Figure 7.6.: Interaction diagram for the part in charge of following the line.

Let's see the mechanism to do this. As you can see in the main scenario diagram, there is an infinite loop that repeats the following procedure forever:

1. The first thing to do is to check if there are pending deliveries. We also check for the robot position, because it is used in almost all cases. An alternate solution is not to obtain the robot position yet, and wait until it is needed.
2. On the one hand, if there are no deliveries:
 - a) The main goal is to arrive to point 0 in order to charge the battery while waiting for new deliveries (we can't let the robot stay in the middle of the corridor). Therefore, if the robot is in point 0, the only thing we need to do is to wait a certain amount of time before going to the beginning of the loop.
3. On the other hand, if there are deliveries:
 - a) First, we thought that it would be a good idea to send a confirmation e-mail to the receiver/sender in order to let him/her know that the robot was approaching. However, for simplicity we decided not to do that and send only one email to the receiver when a delivery is booked.
 - b) We need to make the robot move to the next point.
 - c) As discussed, we need to check whether the robot needs to stop at the point that he approached or not. To do so we will check the most priority delivery. Note that in the computation of priorities the actual position of the robot will be a parameter that increases the priority of the near deliveries.
 - i. First, we will achieve the list of the pending tasks and the current position of the robot.
 - ii. For each task, we will calculate the distance between the position of the robot and the point where the robot should go to fill the pickup/delivery. This value will be d .
 - iii. For each task, we will calculate its priority, following the formula: $p_f = d_{max}/d + p$, where p_f is the final priority (real number between 0 and 2) and p is the priority that was specified by the user (number between 0 and 1). d_{max} is the maximum distance that can be between two points. With this formula the distance is as important as the priority specified by the user.
 - iv. We will return the most priority delivery (the one that have greater p_f).
 - d) If the point of the most priority delivery is the current point, then we need to pick up or deliver the package. Note that, to do so, in the diagram, we compare the position of the delivery with the next position of the robotPosition variable. Note that this works, because the robotPosition variable haven't been updated with the new position, so it is already pointing to the previous point.
 - i. To do so, first we will ring the buzzer (note that the robot is stopped in the position we are interested in).
 - ii. Then, the system starts a time counting.
 - iii. If the user does not appear in a certain amount of time (timeout), the robot will send a message to the server.
 - iv. On the main success scenario, however the user will enter the password and will press enter.
 - v. When this happens, the robot sends the password to the server, which will check it.
 - vi. Just after this, the server will retrieve/update the information about which package is in each secure box section.
 - vii. The server will send a command to the robot to make it open the correct tray.
 - viii. When the user finishes, he/she will press enter in the robot and the server will know that the pick up/delivery was done correctly.

ix. In this case the system will change the state of the concrete delivery. If it was in the state of booked, it will be now in the picked up state. If it was in the picked up state, now it will be in the delivered state.

e) If it is not the most priority, we can just return to the beginning of the loop.

Note that with this solution the robot will remain at the point 0 when there are no deliveries. On the other hand, new deliveries are allowed to be booked meanwhile the robot is moving. Such new deliveries are processed with a really simple, functional and easy-to-maintain algorithm.

Note also that with our approach the robot may be stopped a period of time in each point, waiting for the server to decide if he must stay there or he must continue. However, as the response time of the server will be faster (order of ms), this stop will not be noticeable. Furthermore, in the final implementation, if this is a problem we can just let the robot non stop at the point, but notify the server that it is in that point. Then, when the server knows if the robot must stop in the point or not, notifies the robot again, and he will stop or not. Again, as the response time will be low, the robot will not stop too far from the 'magnet' (the robot will still be in front of the corresponding office).

8. CLASS DIAGRAM AND INTERFACE SPECIFICATION

8.1. Class Diagram

Figure 8.1 illustrates classes and their associations in the project. Each class has some kind of interconnections with one or more other classes. Due to the complexity of the class diagram, we first draw a class diagram with class names only (without their detailed data types and operation signatures). Then we split the big picture into five small pieces, and include detailed class data type and operation signature information in these five parts. Besides, for simplicity, we omitted several insignificant classes such as `TimeoutException`, `NonBookedDeliveryException` in this whole class diagram, and leaved the detailed description of these classes in next section.

From the class diagram we can see that there are mainly two parts. The classes and packages in the first column are classes and packages in the server/client side, and the other column of classes and packages are from the robot-side. The coding of these two will be done separately. From the Server side, total 37 classes consist 3 main layer: presentation, logic, and resource layer. Among these layers, the resource layer can be split into 3 smaller sub-layers in according to their functionality: `resource.communication`, `resource.data`, `resource.datacontroller`. The functionality of each layer is explicit just as the layer's name shows, resources layer works like a database and is mainly used to store and fetch data; logic layer mainly takes charge of controlling and presentation layer is responsible for showing different web interface with customer. The relationship between each layer is also simple: based on database, logic layer manipulates customer's request and communicate with customer through presentation layer. The detailed description of each layer is as follows:

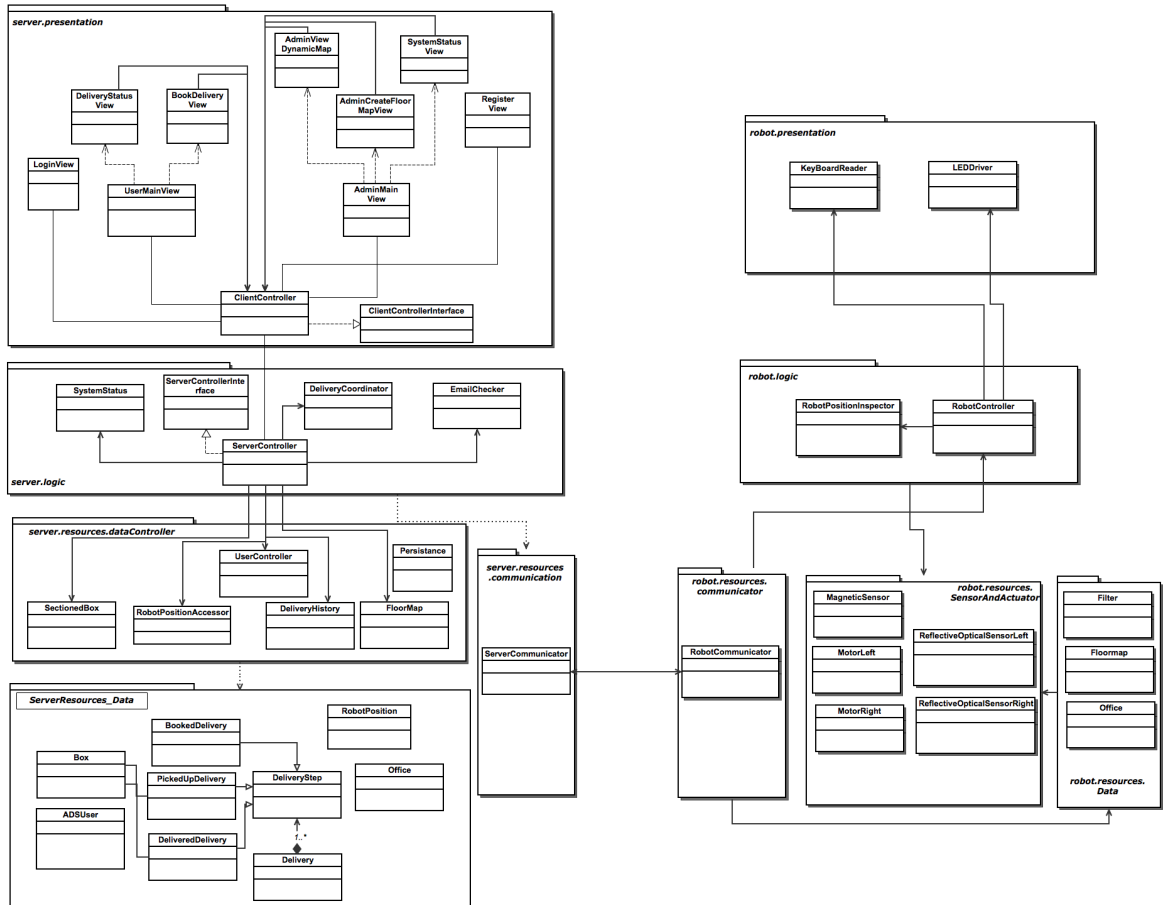
There are 11 classes in the presentation layer, which means these classes take charge of showing different kinds of web interfaces with customer and of implementing the function of communication between the system and the customer. These classes are the `AdminCreateFloorMapView`, `AdminMainView`, `AdminViewDynamicMap`, `BookDeliveryView`, `ClientController`, `ClientControllerInterface`, `DeliveryStatusView`, `LoginView`, `RegisterView`, `SystemStatusView` and `UserMainView`. Among these classes `ClientController` is the main class which implements the Login state conversion of user.

Another layer in this system is the Logic Layer. In this layer algorithms of several Control Unit is implemented. These classes include `DeliveryCoordinator`, `EmailChecker`, `ServerController`, `ServerControllerInterface`, `ServerInitializedException`, `ServerNonInitializedException`, `SystemStatus`. They help implement the flow of how the Central Station received a new delivery order, put it into the queue, dispatch the order, send robot to sender's or receiver's location, password check and notification.

The last layer in the system is the Data Layer. This layer deals with storing and fetching data and encapsulation of data. The classes that store data are `ADSUser`, `BookedDelivery`, `Box`, `DeliveredDelivery`, `Delivery`, `DeliveryStep`, `Office`, `PickedUpDelivery` and `RobotPosition`. By communicating with these classes, the system can record, know, or update any of the data to make a judgment. The classes that is the encapsulation of data include `DeliveryHistory`, `FloorMap`, `Persistence`, `RobotPositionAccessor`, `SectionedBox` and `UserController`. These classes encapsulate previous lower-level data into higher-level structure, such as generating floor map from the `Office`, or generating `DeliveryHistory` from the class `BookedDelivery`, `DeliveredDelivery` and `PickedUpDelivery`. Especially, the class `Persistence` is used to communicate with MySQL database.

From the robot-side, we also have a three-tier architecture: presentation layer, logic layer and resources layer. The presentation layer implements the function of inter-connection with the user, such as ringing the buzzer and record keypad input. The second layer is the logic layer, which handles with moving/stop information from the server and execute instruction to the lower-level of robot actuators to move or

Class Diagram



create and share your own diagrams at gliffy.com



Figure 8.1.: Class diagram

stop. It will also detect current location for robot and report to the server-side. The lowest-level layer is called resources layer, with the same name as that in server-side. This layer contains three packages, communicator, SensorsAndActuator, and data package. Among these three package, communicator is used to communicate with robot communicator. The subpackage SensorsAndActuators contains classes whose names are MagneticSensor, ReflectiveOpticalSensorLeft, ReflectiveOpticalSensorRight, MotorLeft, and MotorRright. These classes functions separately to sense magnetic signal and light signals, or motivating the robot to move. The last subpackage "data" includes classes named Filter, FloopMap and Office. These classes updates information from the sensor and server-side in reference for movement of the robot.

8.2. Data Types and Operation Signatures

The detailed class specification in UML notation of each class is shown in the Partial Class Diagram (Figure 8.2, Figure 8.3, Figure 8.4, Figure 8.5, and Figure 8.6). For simplicity, we will introduce our class functions and operations in the package order. In this whole system, we build three main packages, from the up to down direction, they are presentation package, logic package and data package, in which data package is split to three smaller packages according to their function: data, datacontroller and datacommunication. The meaning of these five packages is explicit as their names. Presentation meanings the interface with users, logic controls the transaction of data and interfaces. We first introduces the first package Logic, and 8 classes (Delivery Coordinator , EmailChecker, Non BookedDeliveryException, ServerController , ServerControllerInterface, ServerInitialized Exception, ServerNonInitialized Exception, SystemStatus) consist of it. The detailed explanation of classes in the logic package is:

1. Delivery Coordinator: Delivery Coordinator works like a coordinator to help with the delivery process. In this class, we will check the validity of sender and receiver by comparing the input from user with the data stored in the database will check the validity of the priority. We also will check whether there are pending deliveries in the system before your current delivery, and the delivery was executed differently during these two situations. Besides, we will check the location of robot when you want to de a delivery, in this way, the robot can pick up and deliver correctly. In a word, the DeliveryCoordinator class is used to helping and controlling the delivery process.
2. EmailChecker: EmailChecker is a simple class and its main purpose is to check the whether the email address the user entered is in a valid email form.
3. NonBookedDeliveryException: This class defines the exception of nonbookdelivery, such as when the sender name is not correct, the priority has a value smaller than 0, which is used to ensure the correct process of booking delivery.
4. ServerController: ServerController works in an important role of controlling the whole system. This class helps us to bind the remote object's stub in the registry. It also works to initialize the whole system, initialize with the testing Data which are inserted to the database by persistence. What's more, it has other important functions, such as checking whether the login is correct, stopping server, registering correctly, booking delivery, searching user and check whether the searching is correct. Also, we can get user delivery list, the system status, user delivery details through this class.
5. ServerControllerInterface: ServerControllerInterface, as the name of it, it is an interface for server. We can create Office and clear office, initialize system, bookdelivery by this interface.
6. ServerInitialized Exception & ServerNonInitialized Exception: These two classes are exceptions, and by properly using exceptions, the whole system can work smoothly. These two classes are used to check in different situations whether the server is initialized or is not initialized.
7. SystemStatus: System Status class is used to detect the position of robot and whether the robot in moving at a specific time.

The Presentation package has 11 classes, and they are AdminCreateFloorMapView, AdminMainView , AdminViewDynamicMap, BookDeliveryView, ClientController, ClientControllerInterface, DeliveryStatusView, SystemStatusView, UserMainView , LoginView, RegisterView

Package Logic

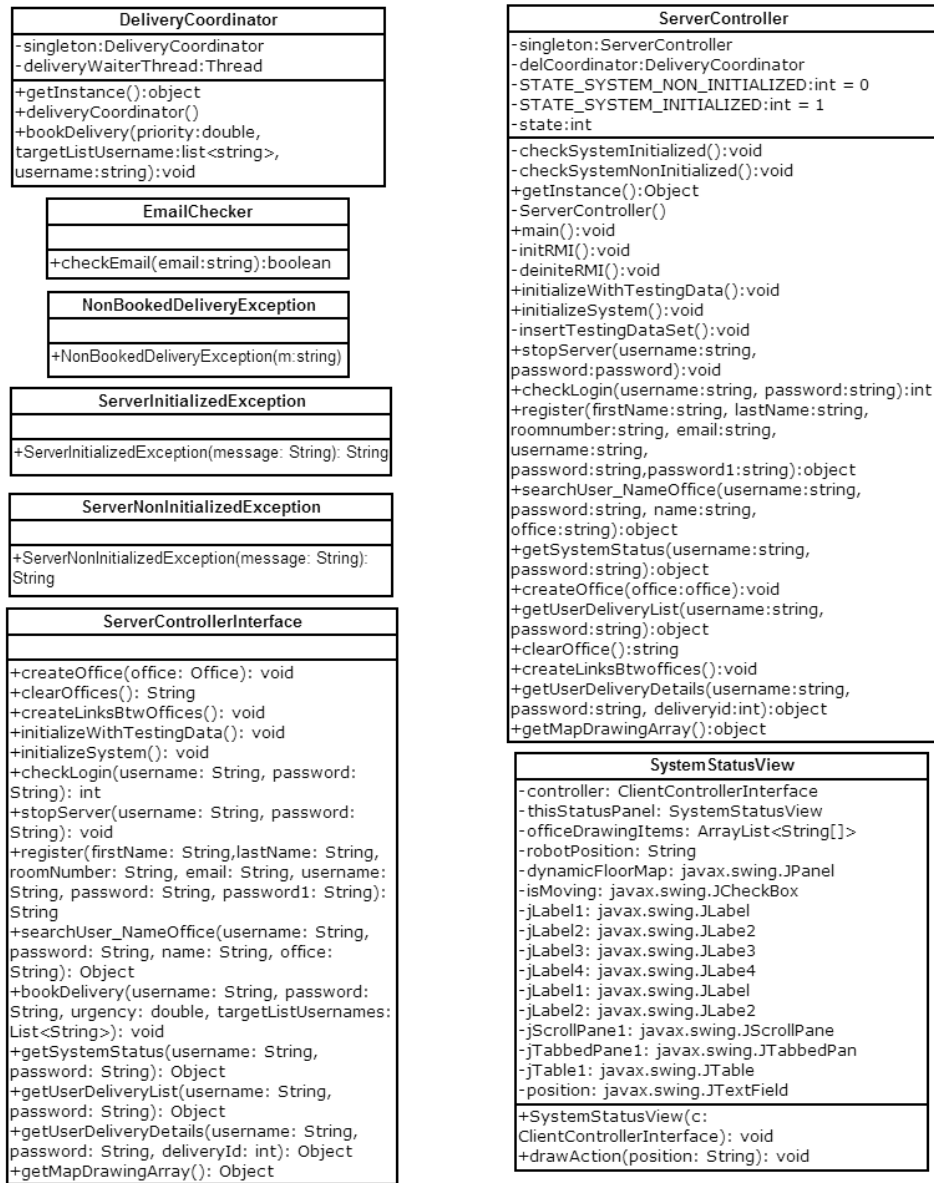
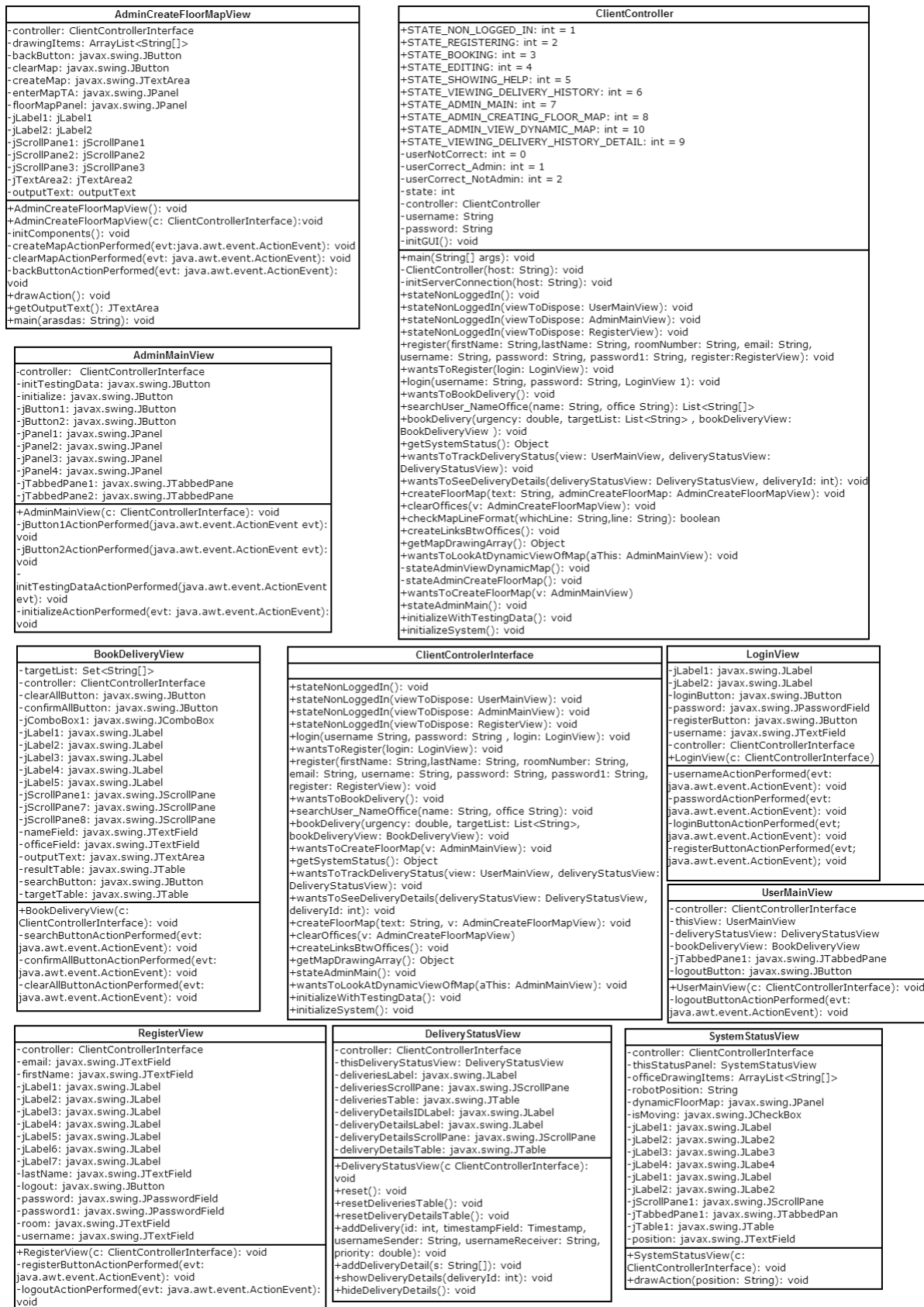
create and share your own diagrams at gliffy.com

Figure 8.2.: Partial class diagram (Part1 of 5): Package'Logic'

Package Presentation



create and share your own diagrams at gliffy.com



Figure 8.3.: Partial class diagram (Part2 of 5): Package 'Presentation'

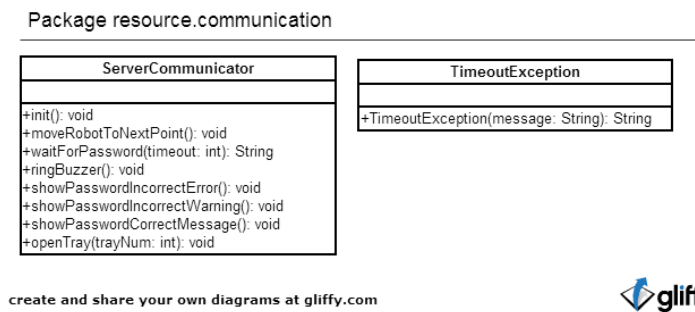


Figure 8.4.: Partial class diagram (Part3 of 5): Package 'resource.communication'

1. AdminCreateFloorMapView: This class implements the user interface between admin and the system when the administrator wants to create a floor map according to current offices location. In this class, we are able to create map, clear map and draw action.
2. AdminMainView: The user is prompted with this interface after login as a administrator. It prompts the administrator with the main interface with options such like dynamic view of map
3. AdminViewDynamicMap: It prompts the administrator with the dynamic location change of the robot.
4. BookDeliveryView : It prompts the user the BookDelivery interface allowing to order deliveries with the GUI. The user is prompted with options of search the receiver by name or office address. The corresponding information will prompt in the bottom blank.
5. ClientController: ClientController is an important class and it runs as a state machine. Totally we have 11 states in this state machine. which are:
 - a) STATE_NON_LOGGED_IN: The client terminal will go into this state when the user has not yet logged in. In this state, the User Interface will show up log-in window, waiting for user to input username and password.
 - b) STATE_REGISTERING: The client terminal will go into this state if the user click the 'register' button in the log-in window. In this state, the system will record personal information that the user has input for registering.
 - c) STATE_BOOKING: The client terminal will go into this state if the user is trying to book a delivery, such as searching for possible recipient and choosing recipients to be delivered.
 - d) STATE_EDITING: The client terminal will go into this state if the user has logged into the system and tries to edit his/her personal information.
 - e) STATE_SHOWING_HELP: The client terminal will go into this state if the user wants to show help and instructions about how this system works. Currently we do not have the implementation of this state because of time limit. However we do reserve this state for future adding up this feature.
 - f) STATE_VIEWING_DELIVERY_HISTORY: The client terminal will go into this state if the regular user has successfully logged in and click the 'Track Status' tab of the main menu. In this state, the system will show a list of delivery history to the user.
 - g) STATE_ADMIN_MAIN: The client terminal will go into this state if the administrator has input his/her administration username and password and successfully logged in.
 - h) STATE_ADMIN_CREATING_FLOOR_MAP: The client terminal will go into this state if the administrator has logged in and tries to create a floor map of a certain office area.
 - i) STATE_VIEWING_DELIVERY_HISTORY_DETAIL: The client terminal will go into this state if one regular user has logged in, clicked the 'Track Status' tab, and double-clicked one specific delivery history item. Detailed delivery history information will show up in the window, such as current delivery status, delivery id, etc.

Package resource.data

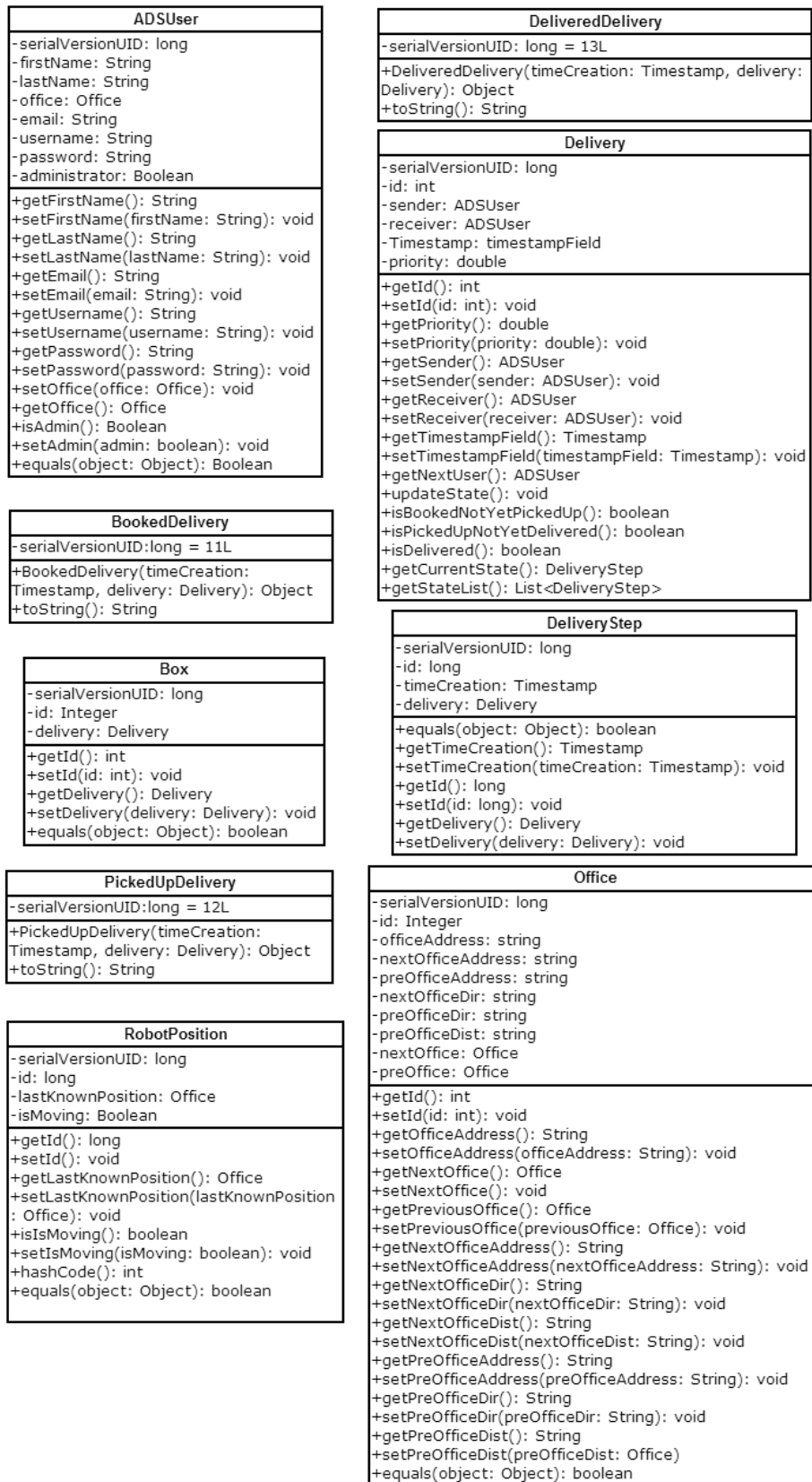


Figure 8.5.: Partial class diagram (Part4 of 5): Package 'resource.data'

Package resource.datacontroller

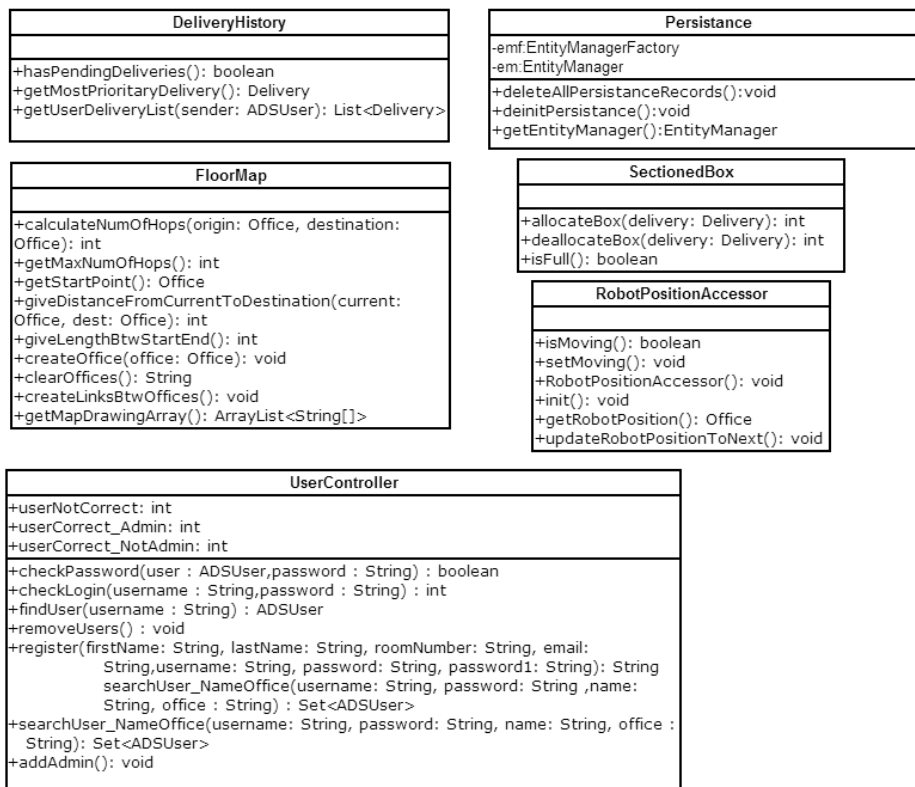
create and share your own diagrams at gliffy.com

Figure 8.6.: Partial class diagram (Part5 of 5): Package 'resource.datacontroller'

- j) STATE_ADMIN_VIEWING_DELIVERY_HISTORY: The client terminal will go into this state if the administrator has logged in, clicked the 'Track Status' tab, then he or she is able to see all delivery history in the system.
- k) STATE_ADMIN_VIEWING_DELIVERY_HISTORY_DETAIL: The client terminal will go into this state if the administrator has logged in, clicked the 'Track Status' tab, and double-clicked one specific delivery history item, then detailed delivery information will show up on the window.

This class helps us to initiate connection with server, change from one state to another state to maintain reasonable interaction with user. In this class, we are able to register, book, edit, searching help and so on.

6. ClientControllerInterface: ClientControllerInterface is the interfaces that prompts users to login, register, enter password to transfer to other states.
7. DeliveryStatusView: This class offers us the operations with delivery status. In this class, we can view the detailed delivery table; also, we can reset, add, and remove delivery details.
8. SystemStatusView: It prompts the users with options of the status of robot and server. The user can get the location or moving status from the interface.
9. UserMainView: The user is prompted with this interface after login. It is the main user interfaces that prompts with options with book deliveries and track status.
10. LoginView: It initiates at the beginning of the program. The user is prompted with the interface with username and password blank to fill in. Also user can click the register button to create new account
11. RegisterView: The user is prompted with this interface after choosing the options of register on the Login interface. The user is prompted with information blanks such as First Name, Last Name, Room, Email, Username, Password, Repeat Password.

The data communication package is consist of ServerCommunicator and TimeoutException.

1. ServerCommunicator: This class is build to help to communicate with the server.
2. TimeoutException: This class is an exception, indicating that the scenario when the user did not entering the correct password in a limited time, the delivery demand cannot be satisfied, this exception ensures that the validity of sender or receiver.

There are 9 classes in the data package, and they are ADSUser, BookedDelivery, Box, DeliveredDelivery, Delivery, DeliveryStep, Office, PickedUpDelivery, RobotPosition, respectively.

1. ADSUser: This class defines the data type of user information (name, password, email, etc) and relative operation in Auto Delivery System.
2. BookedDelivery: This class defines the booked delivery
3. Box: Defining the data structure of mail box of the system.
4. DeliveredDelivery: Defining the delivered delivery.
5. Delivery: Depicting the process and participants of booking and picking a delivery.
6. DeliveryStep: Providing the basic information of a delivery.
7. Office: Defining the data structure of office where is assumed to be the place of receiving mails/- packages.
8. PickedUpDelivery: Defining the picked up Delivery
9. RobotPosition: Defining the data structure of the robot position and the process of tracking the position of the robot.

The datacontroller package is consist of 6 classes, and they are DelieveryHistory, Persistance, FloorMap, Secotionedbox, RobotPostionAccessor, and UserController, respectively.

1. DelieveryHistory: This class mainly used to assist delivery. It's main function is to record the delivery history, to check whether there are pending deliveries, and get the most priority delivery.

2. Persistence: This class is mainly used to communicate with the database and accomplish multi-threading work.
3. FloorMap: This class defines the detail of robot's map, including the start point, the distance from current to destination, the links between each offices, etc.
4. Sectionedbox: This class mainly used to check whether the status of boxes, including whether the box is full or not, allocating and deallocating the box.
5. RobotPositionAccessor: This class is used to detect and set the robot's position status, such as detect whether the robot is moving, getting the current position, and updating the current robot position to the next point.
6. UserController: This UserController class is used to check the authentication of users, including check the password and the login status. it is also used to help user control, such as registering, removing users, finding users, ect.

Next is the packages of the robot-side classes. In this side, we also have three layers as the server system side: they are namely robotLogic layer, robotPresentation layer and robotResource layer. First let's introduce the classes in the robotLogic layer

1. RobotPositionInspector: This class mainly is responsible for tracking robot position and status
2. RobotController: This class mainly distribute the work requested by the central server to robot

Next is the robotPresentation layer. In this layer, we have two classes: keyBoardReader and LEDdriver

1. keyBoardReader: This class mainly gets access to the value of pressed keys in the numerical keyboard
2. LEDDriver: This class mainly operates the LEDs status of the robot

The last layer is the resources layer. We divide this packages into three subpackages namely resources.communication, resources.data. First is the communication subpackages, it contains one class: Robot Communicator

1. Robot Communicator: This class maintains a regular contact from the central server to the robot and allows commands and responses between robot and server.

Second is the data subpackages, it contains 3 classes: the Filter, Floormap and Office.

1. Filter: This class keeps record of recent state of the sensors to avoid noise.
2. Floormap: This class records map data of the office floor.
3. Office: This class restore the data of office where is assumed to be the place of receiving mails/-packages.

In the SensorAndActuator sub-package, we have classes which are named MagneticSensor, ReflectiveOpticalSensorLeft, ReflectiveOpticalSensorRight, MotorLeft and MotorRight

1. MagneticSensor The class MagneticSensor mainly functions as periodically collecting data sensed by the magnetic sensor embedded on the surface of robot. in the robot
2. ReflectiveOpticalSensorLeft This class ReflectiveOpticalSensorLeft mainly functions as periodically sensing the left-side light sensor and collecting the data about whether the robot moves along the left-side white tape. Note that the left-side sensor data collection should synchronize with the data collection with ReflectiveOpticalSensorRight to guarantee the robot is moving along the track.
3. ReflectiveOpticalSensorRight This class ReflectiveOpticalSensorRight mainly functions as periodically sensing the right-side light sensor and collecting the data about whether the robot moves along the right-side white tape. Note that the right-side sensor data collection should synchronize with the data collection with ReflectiveOpticalSensorleft to guarantee the robot is moving along the track.
4. MotorLeft This class execute the actual motivation to drive the left motor of the robot. It will set up robot movement by setting parameter levels of selected pins to drive the left-side motor.
5. MotorRight This class execute the actual motivation to drive the right-side motor of the robot. It will set up robot movement by setting parameter levels of selected pins to drive the left-side motor.

Domain Concept	DeliveryCoordinator	EmailChecker	ServerController	ServercontrollerInterface	SystemStatus	AdminCreateFloorMapView	AdminMainView	AdminViewDynamicMap	BookDeliveryView	ClientController	ClientControllerInterface	DeliveryStatusView	LoginView	RegisterView	SystemStatusView	UserMainView	ServerCommunicator	TimeoutException	ADUser	BookedDelivery	Box	DeliveredDelivery	Delivery	DeliveryStep	Office	PickedUpDelivery	RobotPosition	DeliveryHistory	FloorMap	Persistence	RobotPositionAccessor	SectionalBox	UserController	KeyboardReader	LEDDriver	RobotCommunicator	RobotController	RobotPositionInspector	Filter	SensorReader				
DeliveryHistory																																												
SecureBoxContents																				X		X																						
RobotPosition																											X																	
PendingDeliveries	X																																											
DeliveryCoordinator	X																							X	X																			
Interface				X		X	X	X	X		X	X	X	X	X	X	X																											
EmailNotifier	X																																											
ServerController		X																	X																									
FloorMap																												X	X															
SessionController										X																																		
PasswordChecker																																										X		
UserManager																																										X		
UserList																				X																								
ServerCommunicator						X											X																											
RobotCommunicator																																										X		
RobotController																																										X		
InterfaceRobot																																											X	
TrackFollower																																											X	
Filter																																											X	

Table 8.1.: Traceability Matrix from Domain Concepts to Class Diagram

8.3. Traceability matrix

Table 8.1 indicates how the softwares classes evolved from the domain concepts in the first report. As shown in the table, some concepts will directly map to the corresponding classes. Like the ServerCommunicator it maps to the ServerCommunicator class. In the domain analysis, it's defined as checking the communication status of the server-robot interaction. It remains the same functionality.

For most of the domain concepts we listed, they evolve two or more classes. A simple reason for this is some domain concepts are complexed and less realistic for us to programme within one class. For instance, The domain concept interface has the responsibility of communicating several things to the users. However, when it comes to the class part, we find that this concept should be devided into the following the classes AdminMainView, AdminCreateFloorMapView, BookDeliveryView, SystemStatusView and UserMainView, Login View, Register View, UserMainView and so on. AdminMainView prompts the administrator with the main interface; AdminCreateFloorMapView prompts the administrator with the dynamic location of the robot; BookDeliveryView prompts the user the BookDelivery interface allowing to order deliveries within the GUI; SystemStatusView prompts the users with the status of robot and server; UserMainView is the main user interfaces that prompts with options with book deliveries and track status. These classes specify different aspects of role the Interface plays and it makes the software more well-constructed.

As from the matrix, we also incorporate two domain concepts into one class. For the domain concepts PasswordChecker and UserManager, we first thought they are two different concepts and played different roles to the system. In the domain analysis of report 1, PasswordChecker has the responsibility to check the password introduced when logging in. UserManager has the responsibility of checking user information and then distinguishing the user and administrator. After then, we find that there is no need to separate them into two classes, so we write these two check function into a class called UserController, which designs to check the password and whether it's administrator or not at the same time.

8.4. Design Patterns

In this section we are going to discuss about two design patterns in our system, which are **Singleton Pattern** and **State Pattern**.

8.4.1. Singleton Pattern

Singleton Pattern is a pattern that restricts the instantiation of a class to one object only. In our system, we have many classes using this kind of design pattern for the benefit of coordinating actions across the

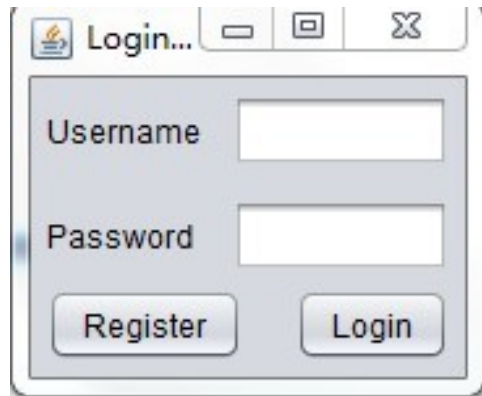


Figure 8.7.: User Interface in state STATE_NON_LOGGED_IN

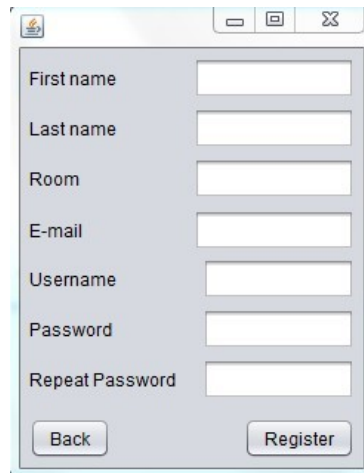


Figure 8.8.: User Interface in state STATE_REGISTERING

whole system.

- in class **DeliveryCoordinator**, we initialize the object **singleton** of this class as a private static member. The constructor of this class `DeliveryCoordinator()` is a private method, so it cannot be called outside this class. Hence the instantiation of this class can be only one object, which is 'singleton' itself.
- in class **ServerController**, we initialize the object **singleton** of this class as a private static member. And with the same from class `DeliveryCoordinator`, the constructor is private, so it cannot be called outside this class.
- In class **ServerCommunicator**, the singleton pattern is also designed the same way, initializing the object of this class as a private static member.

When other classes need to use the object of the class with singleton design pattern, they call public static function `getInstance()` of this class to obtain the instance of this class. It is a good way to reduce coupling and complexity among classes and layers.

8.4.2. State Pattern

We are going to discuss the state pattern in our system, though currently in our Java code, we have not implemented this pattern yet. In the following page we will first show how our state machine is implemented in our system, and then we will propose how this can be improved by using state pattern.

Currently, in the class `ClientController`, we maintain a state machine in charge of communication between server and controller. We have 11 states in total. What we use to distinguish between dif-

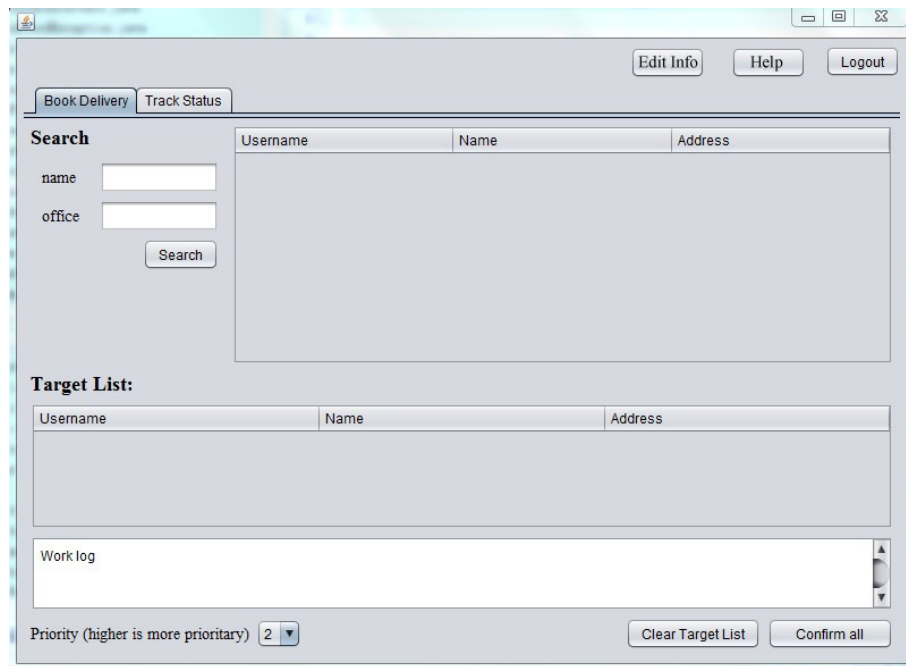


Figure 8.9.: User Interface in state STATE_BOOKING

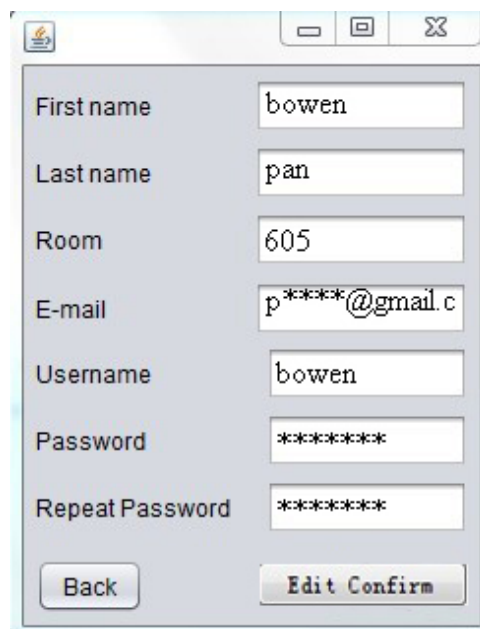


Figure 8.10.: User Interface in state STATE_EDITING

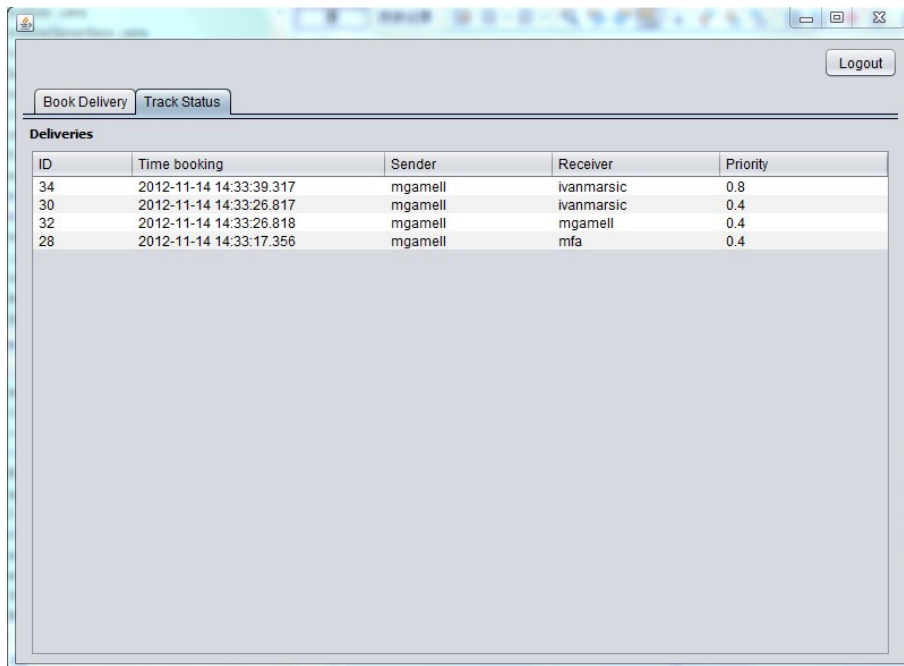


Figure 8.11.: User Interface in state STATE_VIEWING_DELIVERY_HISTORY

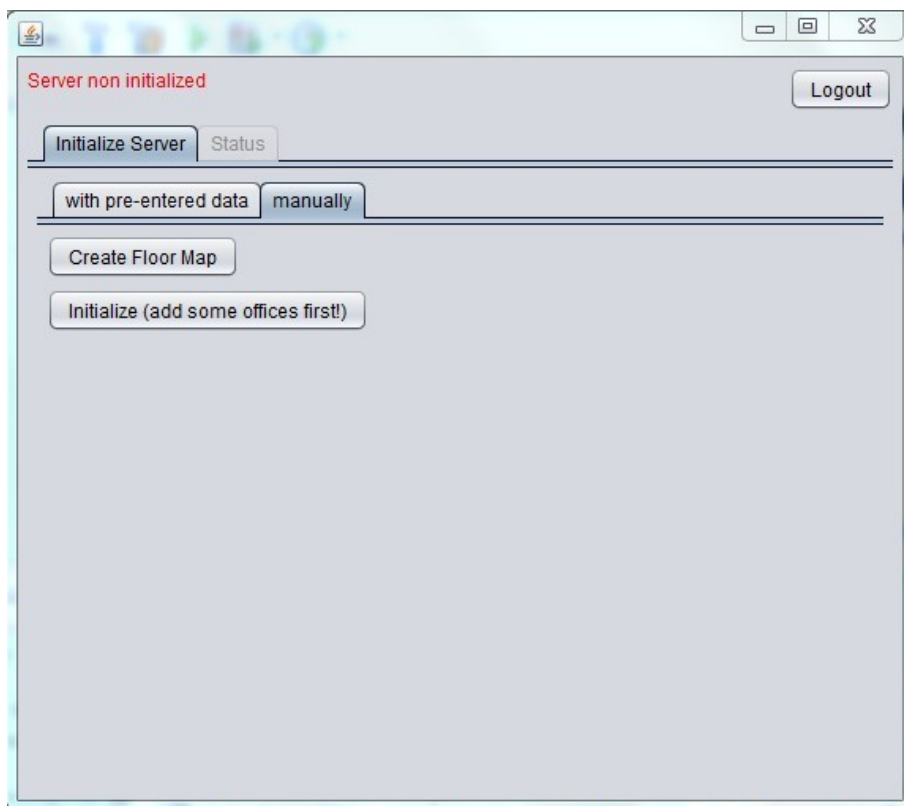


Figure 8.12.: User Interface in state STATE_ADMIN_MAIN

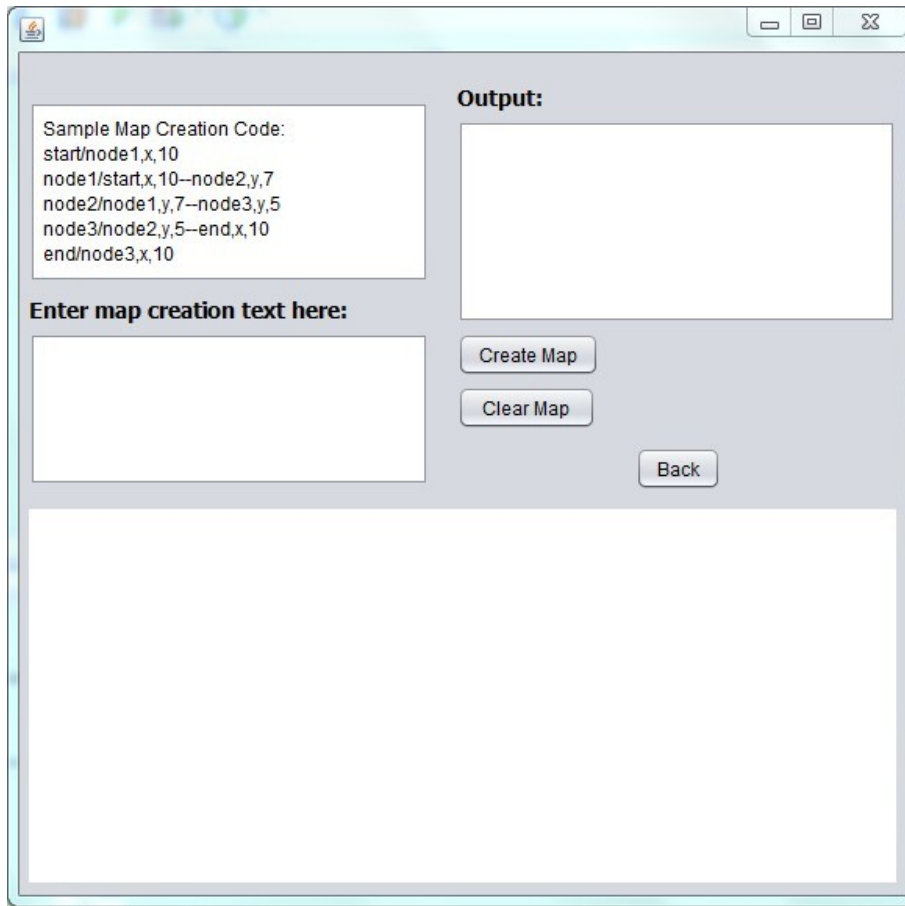


Figure 8.13.: User Interface in state STATE_ADMIN_CREATING_FLOOR_MAP

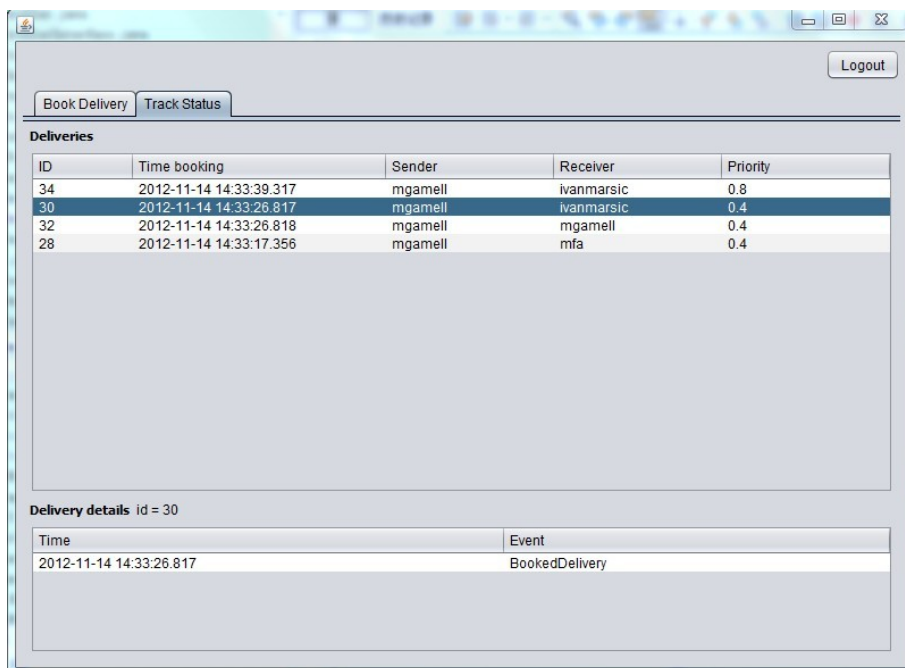


Figure 8.14.: User Interface in state STATE_VIEWING_DELIVERY_HISTORY_DETAIL

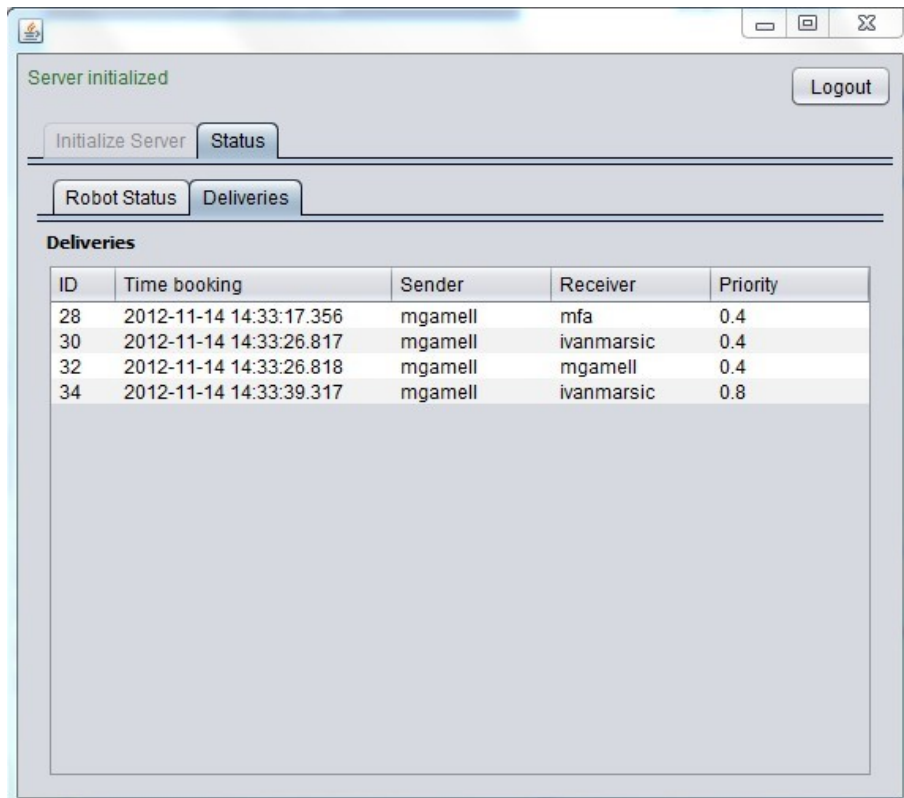


Figure 8.15.: User Interface in state STATE_ADMIN_VIEWING_DELIVERY_HISTORY

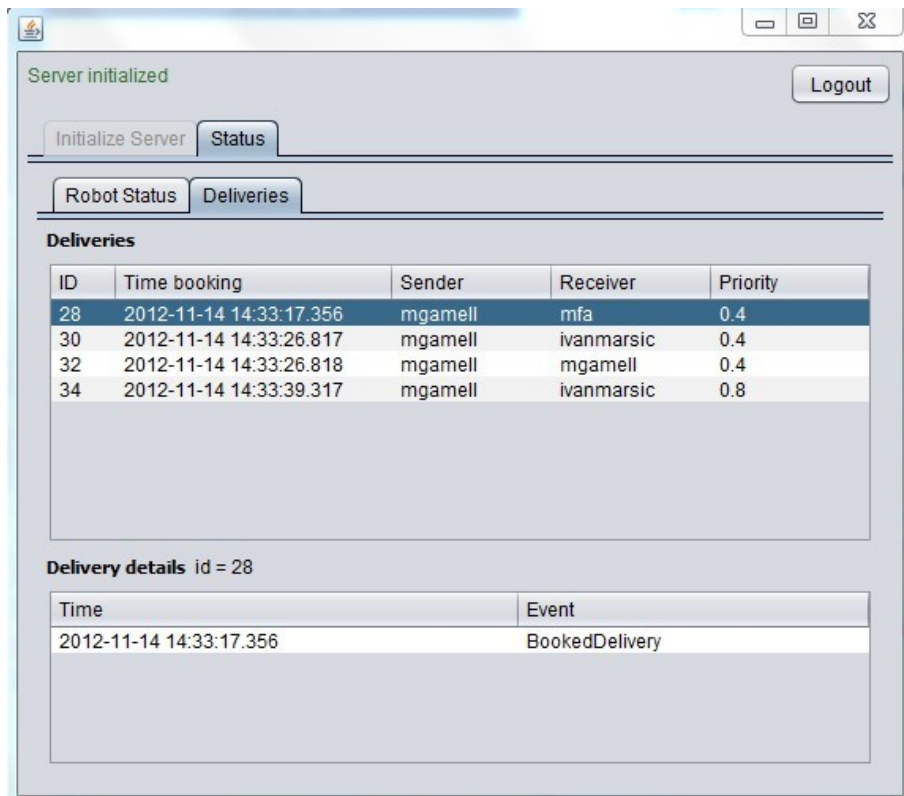


Figure 8.16.: User Interface in state STATE_ADMIN_VIEWING_DELIVERY_HISTORY_DETAIL

ferent states is the final integer value of each state. For example, the final integer value of state `STATE_NON_LOGGED_IN` is 1, and the final integer value of state `STATE_REGISTERING` is 2. Generally speaking, these states control which user interface to show to the user and listening for possible events to change to other states. These states are:

- **STATE_NON_LOGGED_IN:** The user interface corresponds with this state can be seen from figure 8.7. In this state, the client will listen to the event of clicking ‘Register’ or ‘login’ button, and jump to the `STATE_REGISTERING` or `STATE_BOOKING` or `STATE_ADMIN_MAIN` state.
- **STATE_REGISTERING:** The user interface corresponds with this state can be seen from figure 8.8. In this state, the client will listen to the event of clicking ‘Register’ button to go to the state `STATE_BOOKING`, or listen to the event of clicking ‘Back’ button to go to the `STATE_NON_LOGGED_IN` state.
- **STATE_BOOKING:** The user interface corresponds with this state can be seen from figure 8.9. In this state, the client will listen to below events:
 - when the user clicks button ‘Edit Info’, the state machine will go to state `STATE_EDITING`
 - when the user clicks button ‘Logout’, the state machine will go to state `STATE_NON_LOGGED_IN`
 - when the user clicks tab ‘Track Status’, the state machine will go to state `STATE_VIEW_DELIVERY_HISTORY`
- **STATE_EDITING:** The user interface corresponds with this state can be seen from figure 8.10. In this state, the client will listen to the event of clicking button of ‘Edit Confirm’ or ‘Back’, then the state machine will go back to state `STATE_BOOKING`.
- **STATE_SHOWING_HELP:** The user interface and state jump algorithm of this state in left to be finished, because of the time limit of this project.
- **STATE_VIEWING_DELIVERY_HISTORY:** The user interface corresponds with this state can be seen from figure 8.11. In this state, the client will listen to below events:
 - when the user clicks button ‘Logout’, the state machine will jump to state `STATE_NON_LOGGED_IN`.
 - when the user clicks tab ‘Book Delivery’, the state machine will jump to state `STATE_BOOKING`
 - when the user clicks any row of delivery history information, the state machine will jump to state `STATE_VIEWING_DELIVERY_HISTORY_DETAIL`.
- **STATE_ADMIN_MAIN:** The user interface corresponds with this state can be seen from figure 8.12. In this state, the client will listen to below events:
 - when the user clicks button ‘Create Floor Map’, the state machine will jump to state `STATE_ADMIN_CREATING_FLOOR_MAP`
 - when the user clicks button ‘Logout’, the state machine will jump to state `STATE_NON_LOGGED_IN`
- **STATE_ADMIN_CREATING_FLOOR_MAP:** The user interface corresponds with this state can be seen from figure 8.13. In this state, the client will listen to the user clicking button ‘Back’ and jump to state `STATE_ADMIN_MAIN`.
- **STATE_VIEWING_DELIVERY_HISTORY_DETAIL:** The user interface corresponds with this state can be seen from figure 8.14. In this state, the client will listen to below events:
 - when the user clicks button ‘Logout’, the state machine will jump to state `STATE_NON_LOGGED_IN`.
 - when the user clicks tab ‘Book Delivery’, the state machine will jump to state `STATE_BOOKING`
- **STATE_ADMIN_VIEWING_DELIVERY_HISTORY:** The user interface corresponds with this state can be seen from figure 8.15. In this state, the client will listen to below events:

- when the user clicks button ‘Logout’, the state machine will jump to state STATE_ADMIN_MAIN, STATE_NON_LOGGED_IN.
- when the admin clicks any row of delivery history information, the state machine will jump to state STATE_ADMIN_VIEWING_DELIVERY_HISTORY_DETAIL.
- when the admin clicks the button ‘Status’, the state machine will just to state STATE_ADMIN_MAIN
- **STATE_ADMIN_VIEWING_DELIVERY_HISTORY_DETAIL:** The user interface corresponds with this state can be seen from figure 8.16. In this state, the client will listen to below events:
 - when the user clicks button ‘Logout’, the state machine will jump to state STATE_ADMIN_MAIN STATE_NON_LOGGED_IN.
 - when the admin clicks the button ‘Status’, the state machine will just to state STATE_ADMIN_MAIN

From the above description we can see that the state machine is kind of complicated, and the class ‘ClientController’ which contains these states are really big. We could find better ways of handling with this complicated state machine jumping algorithm. One of the improvement may be using **state design pattern**.

State design pattern can reduce the coupling of difference states in one big class. What we can do is to separate each state as a class, and these 11 classes will all implement the interface, which could be named as ‘ClientStateMachineInterface’. This interface well encapsulate the complex algorithms in this state machine. Whenever an event is triggered, what the object of user interface (which can be regarded as Context) need to do is to call the method handle() on its current state. That is to say, the object of user interface only need to provide current state to the ClientStateMachineInterface whenever it receives a request() call to handle some certain events. This would be much easier to maintain and when we change the contents of some certain state, contents of other states don’t need to change.

8.5. Object Constraint Language (OCL)

The following are the OCL for each class:

CLASS DeliveryCoordinator:

1. Invariants: there are some deliveries need to be coordinated
context DeliveryCoordinator inv:
self.thread>0
2. Pre-conditions: Must have data about delivery process stored in the data base
context DeliveryCoordinator : Boolean
pre: self.storeddataavailable == true
3. Post-conditions: is empty/input from user and the data stored in the database is mismatch.
context DeliveryCoordinator : Boolean
post: self.storeddataavailable == false

CLASS EmailChecker

1. Invariants: the email address the user entered must be in a valid email form
2. Pre-conditions:the user must enter an email address
context EmailChecker: Boolean
pre: self.emailentered == true
3. Post-conditions: no email address input/the input email address is invalid
context EmailChecker: Boolean
post: self.emailentered == false

CLASS NonBookedDeliveryException

1. Invariants: the delivery information must be corrected

2. Pre-conditions: Must have data about delivery process stored in the data base
context NonBookedDeliveryException: Boolean
pre: self.storeddataavailable == true
3. Post-conditions: is empty/input from user and the data stored in the database is mismatch.
context NonBookedDeliveryException: Boolean
post: self.storeddataavailable == false

CLASS ServerController

1. Invariants: the system can be initiated
context ServerController inv: self.STATE_SYSTEM_INITIATED==1
2. Pre-conditions: the remote object's stub must be bound in the registry
context ServerController : Boolean
pre: self.stubbound == true
3. Post-conditions:the system cannot be initiated or controlled
post: self.STATE.SYSTEM_INITIATED==0 self.ServerController==NULL

CLASS ServerControllerInterface

1. Invariants: N/A
2. Pre-conditions: N/A
3. Post-conditions: N/A

CLASS ServerInitializedException

1. Invariants: the system must check in different situations whether the server is initialized or is not initialized
context ServerInitializedException inv: self.statuschecked==true
2. Pre-conditions: the system can be initiated
context ServerInitializedException:
pre: self.STATE.SYSTEM_INITIATED==1
3. Post-conditions:the system cannot be initiated/cannot check the exception
context ServerInitializedException:
post: self.STATE.SYSTEM_INITIATED==0 self.ServerController==NULL

CLASS ServerNonInitializedException

1. Invariants: the system must check in different situations whether the server is initialized or is not initialized
context ServerController inv: self.statuschecked==true
2. Pre-conditions: the system can be initiated
context ServerNonInitializedException:
pre: self.STATE.SYSTEM_INITIATED==1
3. Post-conditions:the system cannot be initiated/cannot check the exception
context ServerNonInitializedException:
post: self.STATE.SYSTEM_INITIATED==0 self.ServerController==NULL

CLASS SystemStatus

1. Invariants: the system must detect if the robot is moving or not
context SystemStatus inv: self.ismoving =NULL
2. Pre-conditions: the system can detect the position of the robot
context SystemStatus: Boolean
pre: self.PositonDetected==true
3. Post-conditions: the system cannot detect the position of the robot
context SystemStatus: Boolean
post: self.PositonDetected==false

CLASS AdminCreateFloorMapView

1. Invariants: the administrator can create floor map according to current offices location
context AdminCreateFloorMapView inv: self.ArrayList =NULL
2. Pre-conditions: the system can detect current position and can be initiated
context ServerController : Boolean
pre: self.PositonDetected==true self.STATE.SYSTEM_INITIATED==1
3. Post-conditions: the system cannot detect current position/cannot initiated
context AdminCreateFloorMapView:
post: self.PositonDetected==false self.STATE.SYSTEM_INITIATED==0

CLASS AdminMainView

1. Invariants: the user can use administrator interface if log in as administrator
context AdminMainView inv: self.ArrayList =NULL
2. Pre-conditions: the user can log into the system
context AdminMainView: Boolean
pre: self.LogIn==true
3. Post-conditions: the user cannot log into the system
context AdminMainView: Boolean
post: self.LogIn==false

CLASS AdminViewDynamicMap

1. Invariants: the administrator can view the dynamic location change of the robot
context AdminViewDynamicMap inv: self.dynamicview==true
2. Pre-conditions: the remote object's stub must be bound in the registry and the system can be initiated
context AdminViewDynamicMap:
pre: self.stubbound == true self.STATE.SYSTEM_INITIATED==1
3. Post-conditions: the system cannot detect current position of the robot/cannot initiated
context AdminViewDynamicMap:
post: self.stubbound == false self.STATE.SYSTEM_INITIATED==0

CLASS BookDeliveryView

1. Invariants: the user can book a delivery after logging into the system
context AdminMainView inv: self.ClientControllerInterface==1
2. Pre-conditions: the user can log into the system
context AdminMainView: Boolean
pre: self.LogIn==true
3. Post-conditions: the user cannot log into the system
context AdminMainView:
post: self.LogIn ==false self.ClientControllerInterface==0

CLASS ClientController

1. Invariants:the system can initiate server connection
2. Pre-conditions: the system can be initiated
context ClientController: int self.STATE.SYSTEM_INITIATED==1
3. Post-conditions: the system cannot be initiated
context ClientController: int self.STATE.SYSTEM_INITIATED==0

CLASS ClientControllerInterface

1. Invariants: N/A
2. Pre-conditions: the system can initiate server connection
3. Post-conditions: the system cannot initiate server connection

CLASS DeliveryStatusView

1. Invariants:the system can initiate server connection

2. Pre-conditions: the system can be initiated
context ClientController: int self.STATE_SYSTEM_INITIATED==1
3. Post-conditions: the system cannot be initiated
context ClientController: int self.STATE_SYSTEM_INITIATED==0

CLASS SystemStatusView

1. Invariants: the system must prompt the users with options of the status of robot and server.
2. Pre-conditions: the user can log into the system
context SystemStatusView: Boolean
pre: self.LogIn==true
3. Post-conditions: the user cannot log into the system
context SystemStatusView: Boolean
post: self.LogIn==false

CLASS UserMainView

1. Invariants: user is prompted with main view interface after login
2. Pre-conditions: the user can log into the system
context UserMainView: Boolean
pre: self.LogIn==true
3. Post-conditions: the user cannot log into the system
context UserMainView: Boolean
post: self.LogIn==false

CLASS LoginView

1. Invariants: N/A
2. Pre-conditions: N/A
3. Post-conditions: N/A

CLASS RegisterView

1. Invariants: N/A
2. Pre-conditions: N/A
3. Post-conditions: N/A

CLASS RobotStub

1. Invariants: the system must provide clear view of the robot's motion during the delivery process
2. Pre-conditions: the remote object's stub must be bound in the registry and the system can be initiated
context RobotStub:
pre: self.stubbound == true self.STATE_SYSTEM_INITIATED==1
3. Post-conditions: the system cannot detect current position of the robot/cannot initiated
context RobotStub:
post: self.stubbound == false self.STATE_SYSTEM_INITIATED==0

CLASS ServerCommunicator

1. Invariants: N/A
2. Pre-conditions: N/A
3. Post-conditions: N/A

CLASS TimeoutException

1. Invariants: Must indicate the scenario when user enter mismatched user name and password 3 times.
context AdminMainView inv: self.InvalidEnter==3

2. Pre-conditions: Must have data about user information correctly stored in the database
context DeliveryCoordinator : Boolean
pre: self.storredatacorrect == true
3. Post-conditions: database is empty/the stored user information is incorrect
context DeliveryCoordinator : Boolean
post: self.storredatacorrect == false

CLASS ADSUser

1. Invariants: Must be a registered user with user name, password, address, email, etc.
2. Pre-conditions: Must have registered before
context ADSUser: Boolean
pre: self.registered_customer ==true
3. Post-conditions: haven't registered before
context ADSUser: Boolean
post: self.registered_customer ==false

CLASS BookedDelivery

1. Invariants: Must have a serial number of booking
context BookedDelivery inv: self.SerialVertionUID==true
2. Pre-conditions: N/A
3. Post-conditions: N/A

CLASS Box

1. Invariants: Must have space available
context Box inv: self.box = FULL
2. Pre-conditions: Must not be full.
context Box: int self.BoxSpaceAvailable >0
3. Post-conditions: Is occupied/has less spots available.
context Box: int self.BoxSpaceAvailable == 0

9. SYSTEM ARCHITECTURE AND SYSTEM DESIGN

9.1. Architectural Styles and Subsystems

As our application is kind of complex in terms of architecture, we use several architectural styles together modeling the system. In this section we will begin discussing the architectural styles of our system from a higher level and then decompose it into several pieces and discuss one by one.

As is shown in figure 9.1, we can generally identify the structure of our system – it is composed of three main parts:

- the client side
- the server side
- the robot

9.1.1. Client/Server and Master/Slave Architectures

From this section, we will discuss about architectural styles in a more specific way. Here we categorize the communication mechanism among the three subsystems (client, server, and robot) into two architectural styles:

- **Client/Server Architecture:** In order to set up communication between the client and the server, we are using a typical **client/server** architecture. This architecture aims at allowing access for one or more clients to several resources or functionalities in a central server. Please note that a client here refers to a Java application in a PC, and multiple clients are allowed to communicate with the central server at the same time. Another proposal of communication architecture could be some parallelized shared architecture (such as P2P). However we are not using this parallelized shared architecture, because the advantages that this kind of architectures can provide, such as scalability, are not critical in our system. Also, a client/server architecture works more stable and is far easier to implement.
- **Master/Slave Architecture:** We have defined a **master/slave** architecture [39] [40] to set up communication between the server and the robot. Please note that this architecture is different from master/worker architecture [41], which is used to parallelize allocating workload, such as Map-Reduce. In our system the robot will wait for commands sent from the server, execute each command and finally return the result, or notify error, to the server.

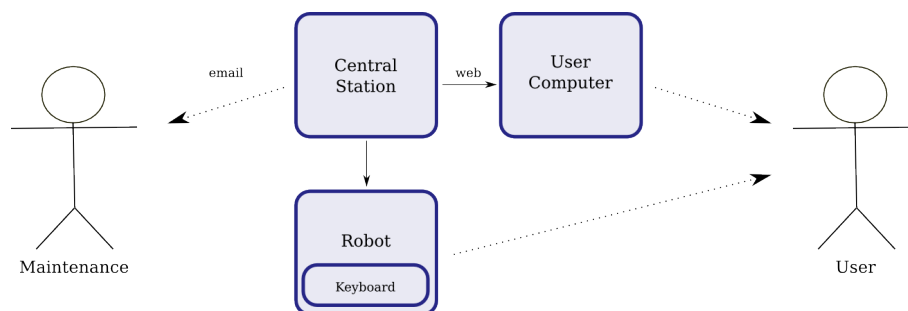


Figure 9.1.: Overview of the system

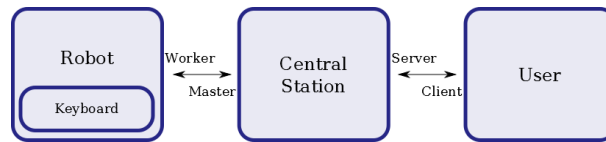


Figure 9.2.: Mapping of the client/server and master/slave architecture styles in our system.

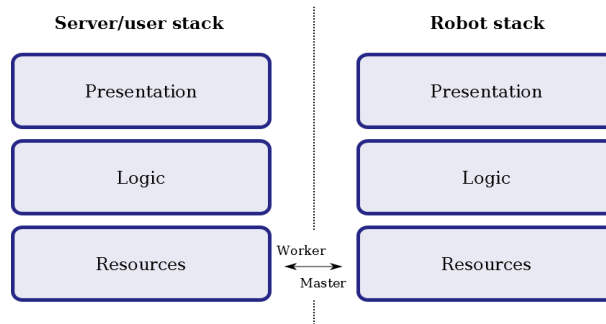


Figure 9.3.: Mapping of the 3-tier architectural style in our system.

As we can see, our approach (see figure 9.2) can support multiple clients and, in the future, it will be easier to add more robots into our system.

9.1.2. Three-tier architecture

From the perspective of entire system, i.e. from both the server part and the robot part, we would like to refine our design by separating the modules of user interface, data processing, and data storage (as well as communication logic) and make them run individually, thus changing one part won't have great influence on other function modules, which is good for reducing system complexity. Therefore, we are using a two-stack 3-tier architecture (figure 9.3). The 3-tier architecture [50] is one kind of N-tier architecture, and is well suited to separate the roles of three components of the system: the user interface (presentation layer), the logic of the system (business rules, logic tier) and the data (typically data tier). The advantage of this structure includes the fact that each layer communicates only with its immediate surroundings (enabling the minimal coupling of software [49]), and the fact that the system is inherently modular [42], eliminating many disadvantages of monolithic software, such as having difficult maintaining the system [43].

In our system, we have instantiated this pattern. We also extend the meaning of the *data* layer into the *resources* layer, with broader definitions compared with *data* layer. The *resources* layer in server side include several submodules: data, dataController and communications, as is shown in figure 9.4. This layer handles with data structure encapsulation, communication with robot part and client, as well as storing data. The *resources* layer in the robot side also include several submodules: communication, sensorsAndActuators and data. The last subpackage of these three, data, will not be in charge of maintaining a redundant copy of the data identical to data submodule in the server. It will store only the information needed for the emergency autonomous positioning of the robot in case the communication between robot and server fails. The sensorsAndActuators are the software submodule that directly monitor the change of sensors (magnet sensors and optical sensors), and that drive the motor to move or stop.

The interface between user and system in the 'client' side would be the Java application running in the form of a window. Interface between user and system in the 'robot' side are the LEDs and the keyboard located on the surface of the robot. They are used as basic authorization and communication between the user and the system. The submodules LEDDriver and KeyboardReader, who are responsible of controlling these two interfaces, will be included in the presentation layer in the 'robot' side.

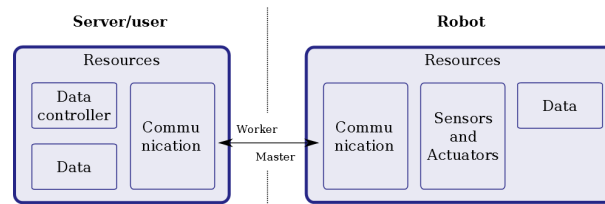


Figure 9.4.: Detail of the resources tier in our system.

9.1.3. Facade Pattern

Now let's focus on the logic layers of server and robot. These layers are the cores of the system, in the sense that they implement the business rules of our system. In order to simplify the interaction between other layers and logic layers, we use a kind of design pattern, which is the Facade pattern.

Facade pattern means 'the face of the building'. This pattern can be explained as 'people walking in the road have no knowledge of what is inside, such as the wiring, the pipeline, etc. What they can see is only the 'glass face of the building' [30]. In Java programming, this pattern can be implemented by using an interface to show the 'glass face' of one or several classes, so that other classes can only see the interface, hiding all complexities of these classes. For example, in the server part, we defined an interface `ServerControllerInterface` in logic layer. This interface displays a friendly face of all methods in class `ServerController`. Thus, in the presentation layer of server, we can use an object of this interface in the class `ClientController`, to call methods in `ServerController`, without too much coupling of these two classes. This pattern aims at making a module easier to use, more readable, so that it can reduce the dependencies of the internal structures of the module.

Note that we have not used the Facade pattern in the resources layer (which is the bottom layer of our three-layer structure) or any sub-packages, because the interface would become unreadable and many different operations would be mixed without any sense.

9.1.4. Event-driven Architecture

We are also using event-driven architecture in the 'robot' part. Event-driven architecture is a software architecture pattern promoting the production, detection, consumption of, and reaction to events [29]. This architectural pattern typically consists of event emitters, and event consumers. Event emitters are those parts in a system which could trigger some kinds of events, and event consumers are those part which are capable of react to certain events.

In our design, the event emitters would be the sensors (including magnet sensors, and optical sensors), the timer, and keyboard on the surface of the robot, as well as the 'filter' that can correctly sense this event happening. Event consumers are mainly the robot-controller, and server-controller in the server part, whose responsibility is to react according to these events. For example, when the magnet sensor detects a magnet was near the robot, the event 'robot approaching the next office location' is triggered, then the robot-controller will react as sending stopping instructions to motors, and communicate with robot to decide whether to wait at this point to pick up package/delivery package or move on to the next point (we assume this point is not the 'home' point where the robot will stay at this point if no pending deliveries)

9.1.5. Summary

To sum up, the architectural patterns and styles used are listed below:

- Client/Server Architecture
- Master/Slave Architecture
- 3-tier Architecture
- Facade Pattern

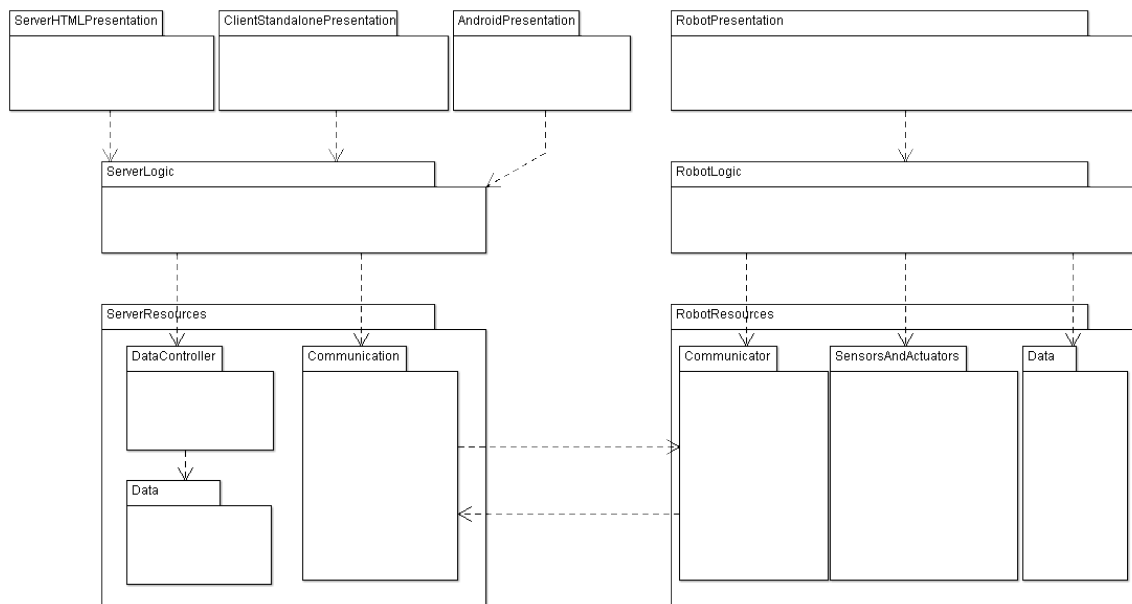


Figure 9.5.: The subsystems of our system. UML package diagram.

- Event-Driven Architecture

The combination of these architectural patterns makes our system easy to understand, build and maintain.

9.2. Subsystems

In the previous section we have already discussed the subsystems that our system have. In figure 9.5 we can see the package diagram including a package for each subsystem.

9.3. Mapping subsystems to hardware

As is shown in previous sections and figure 9.5, the subsystems can be easily mapped into the hardware components:

- The ServerLogic and ServerResources packages are allocated in the central server.
- The RobotPresentation, RobotLogic and RobotResources packages are allocated in the robot (Arduino board).
- The ServerHTMLPresentation is allocated in the server, and the user needs a web browser to access this presentation. Please note that we are not going to implement this feature in this semester due to time limitation. The ServerHTMLPresentation will be considered as one part of future work.
- ClientStandalonePresentation package is allocated in client's PC.
- Finally, the AndroidPresentation is allocated in the Android-based smartphone in the form of app. Please note that we are not going to implement this feature in this semester because of time limitation. The Android client app will be considered as one part of future work.



Figure 9.6.: The persistent entities

9.4. Persistent Data Storage

As we want our system to be fault tolerant, we need to make sure data information can be stored and updated in an appropriate pattern:

- The system needs to store all the information regarding current and past deliveries or users in a persistent storage
- all the operations involving the information must operate using transactions:
 - at low level accesses: when updating a single value from the storage, all modern relational SQL databases allow transactionality
 - at higher level: we only want to update the information about a delivery if, and only if, the designated delivery information to be updated has already finished.

The easiest way to do it is using a relational database, such as MySQL and store all the information in form of tables.

Therefore, we will need to store the classes in the `ServerResources.Data` subpackage: `ADSUser` (We abbreviate ‘Users in Automatic Delivery System’ as ‘ADSUser’), `Box`, `Delivery`, `DeliveryStep` (which will represent also its subclasses `BookedDelivery`, `PickedUpDelivery` and `DeliveredDelivery`), `Office` and `RobotPosition`. This is represented in figure 9.6. Figures 9.7, 9.8, 9.9, 9.10, 9.11 and 9.12 represent the fields we need to store for each class. The classes can be linked together by sharing some common fields. For example, the classes `Box`, `Delivery`, `DeliveryStep`, `Office` and `RobotPosition` all have the field ‘ID’, thus by inquiring the ID of a specific delivery, we can know information of all these fields. Another example is that `Delivery` class has the fields `RECEIVER_USERNAME` and `SENDER_USERNAME`, so by inquiring these user names in class `ADSUser`, the system can get the user information according to the names in the `RECEIVER_USERNAME` and `SENDER_USERNAME` fields.

In order to access the DBMS we will not use direct low-level SQL instructions, as many programs used to use. On the contrary, we will use much modern techniques that delegate the persistence of the classes to dedicated libraries. In our case, as we will develop in Java, we will use the subset of the Java Enterprise Edition [36] in charge of data persistence: Java Persistence API (or JPA) [37]. The second version of this API (JPA2, which is implemented in libraries such as EclipseLink [28], which can be used in regular Java Standard Edition applications) allows the addition of Java annotations [35] in the classes that the programmer wants to be automatically persisted (see [38]). The most relevant annotations include `@Entity` (to define a persistent class), `@Id` (to define the primary key), `@OneToOne`, `@ManyToOne` and `@OneToMany` (to define relationships between persistent entities).

9.5. Network Protocol

Our system has three main components: server part, client part and robot part. All of these parts have one class as an abstraction interface responsible for the communication (as can be seen in the class diagram part). In this scheme, there are two communication channels necessary for the operation of the system. One of them is the channel between the server side and the client side and the other one is the channel between the server side and the robot side (see figure 9.13).

Table Name: Schema: **ads**

Collation: Engine:

Comments:

Columns

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
USERNAME	VARCHAR(255)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
ADMINISTRATOR	TINYINT(1)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	'0'
EMAIL	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
FIRSTNAME	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
LASTNAME	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
PASSWORD	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
OFFICE_ID	INT(11)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Figure 9.7.: 'ADSUser' persistent class. Detailed field list.

Table Name: Schema: **ads**

Collation: Engine:

Comments:

Columns

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
ID	INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
DELIVERY_ID	INT(11)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Figure 9.8.: 'Box' persistent class. Detailed field list.

Table Name: Schema: **ads**

Collation: Engine:

Comments:

Columns

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
ID	INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
PRIORITY	DOUBLE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
TIMESTAMPFIELD	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
RECEIVER_USERNAME	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
SENDER_USERNAME	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Figure 9.9.: 'Delivery' persistent class. Detailed field list.

Table Name: Schema: **ads**

Collation: Engine:

Comments:

Columns

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
ID	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
DTYPE	VARCHAR(31)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
TIMECREATION	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
DELIVERY_ID	INT(11)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Figure 9.10.: 'DeliveryStep' persistent class. Detailed field list.

Table Name: Schema: **ads**

Collation: Engine:

Comments:

Columns

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
ID	INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
NEXTOFFICEDIR	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
NEXTOFFICEDIST	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
OFFICEADDRESS	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
NEXTOFFICE_ID	INT(11)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
PREOFFICE_ID	INT(11)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Figure 9.11.: 'Office' persistent class. Detailed field list.

Table Name: Schema: **ads**

Collation: Engine:

Comments:

Columns

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
ID	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
ISMOVING	TINYINT(1)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	'0'
LASTKNOWNPOSITION_ID	INT(11)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Figure 9.12.: 'RobotPosition' persistent class. Detailed field list.

9.5.1. Network between Server and Clients

As the communication protocol between the Server and the Client side, we chose to use Java RMI. The main reason of this choice of ours is that Java RMI allows any Java Object type to be used and also it allows both the client and server to dynamically load new object types as required (reference [33]) while other protocols and simple Java sockets allow only simple data types or defined structures to be used as the communication data. In that sense, Java RMI is more powerful than any other. Also positive feedbacks from the users of Java RMI API are also important factors for us to choose it as network communication protocol between Server part and client part.

Using RMI enables us, the developers, to spend our time on thinking about the core functionalities of the system rather than doing some further implementations to make communication on primitive data types and to make the communication robust. In brief, RMI introduces wide freedom to the developers in terms of design and implementation decisions.

9.5.2. Network between Server and Robot

We use a WiFi shield (reference [22]) in the robot side to connect the robot side with the internet through WiFi. Then we will be able to control the robot part by simply accessing the client over the internet from the server part. The access point providing the internet for this communication can be a wireless access point already installed and used by people in the working environment.

Since the communication data between the server and the robot will not be Java objects (as is the case for the communication between the server and the client side) but simply primitive data types, such as ‘strings’, and also since we do not have any other important communication considerations (e.g. energy efficiency, low interference etc.) and constraints (i.e. communication channel is indoor so there are no difficulties related with outdoor wireless communication such as line of sight, fading etc.), we simply use Internet layer sockets to function as interface to communicate between robot and server.

As one can notice during the explanation above, we did not make any thinking about the security of the communication channel between the server and the robot. The reason of this attitude of us is that our system-to-be does not need a more secured channel than that of provided by the 802.11 standards (reference [23]). The wireless access point will be accessed by both the server and the robot by using simple WEP based authentication as the other users. The rest will be taken care of 802.11 standards.

9.6. Global Control Flow

In this part, some important aspects about the control of the system will be introduced.

9.6.1. Execution Orderness

- In Server Side

Both client in the user PC and the robot can be regarded as ‘users’ of the server. Hence, server provides some functionalities for these two users and these functionalities are executed based on the requests done by the users. Server side provides an interface to its users and waiting for its users to request some functionalities.

- In Client Side

The user of this side is the people who are using the system. This side is event-driven such that once the user launches this side of the system, he/she is forced to give some desired inputs to the system generating the corresponding events (e.g. mouse clicks to the user interface buttons and guided areas) and client side changes its state according to these events and their results. The detailed state diagram explaining the event-operation of the client side can be understood (figure 9.14).

- In Robot Side

The tasks of this part are done in an event-driven manner and the events are generated by the server side. Main functions of this part are movement of the robot to a particular place, hand in and pick up packages related operations.

9.6.2. Time Dependency

There is only one time dependency in our system which is in the robot side. After the robot reaches to the office for pick up or hand in operations a timer will be started and the buzzer will start to ring, to notify the sender or receiver for a fixed amount of time. When it reaches timeout, the robot will postpone this picking up package or delivering package to a later time and continues with other tasks.

9.6.3. Concurrency

The server side of our system is able to provide services to multiple clients concurrently. This is achieved by the multi-threading mechanism provided by the API Java RMI (Remote Method Invocation). Java RMI provides a kind of policy which can be described as follows: In the server part, calls originating from different clients Virtual Machine will execute in different threads. From the same client machine it is not guaranteed that each method will run in a separate thread [44].

9.7. Hardware and Software Requirements

In this part, the resources that the system depends upon will be presented.

- Server Side Requirements

In the server part, system needs a server computer for providing the main functionalities of the system. For an average working environment, our system needs only an average machine which can run mysql database, java seamlessly and have wifi capability. In that sense, an average computer in the market would be enough to serve as a server to our system. So we can summarize the requirements of the server machine as follows;

1. Mysql database should be installed
2. Java Runtime Environment (JRE) should be installed
3. Wifi capability should be present on the machine (i.e. necessary for the communication between the server and the robot as explained in part e).

- Client Side Requirements

Users of the system will be able to use the functionalities provided by a simple java interface. In that sense, any average computer be a proper machine for the client side purposes. So for the client side part of the program to run correctly, the user machine should have the following requirements;

1. Java Runtime Environment (JRE) should be installed
2. Internet access should be present

- Robot Side Requirements

What we call as “Robot” is the delivery machine itself and we designed it to be as cheap and as efficient as possible to accomplish the system-to-be’s main functionalities. We will use one of the most popular board microcontroller family, arduino. In that sense, our delivery machine will be a simple example of mobile robot which is controlled by the arduino microcontroller. For the robot side to be able to accomplish the functionalities which are explained previously, the following requirements should be present;

1. Arduino microcontroller board (reference [3])
2. Numeric keypad (reference [21]) (i.e. for the user to interact with the system)
3. Wifi Shield (reference [22]) (i.e. to create the communication channel between the server and the robot side; look part e for more explanation)

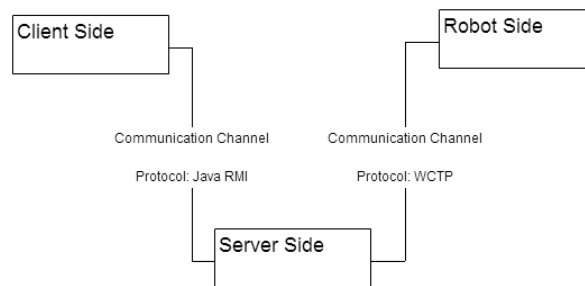


Figure 9.13.: Overview of the communication channels between the main system components

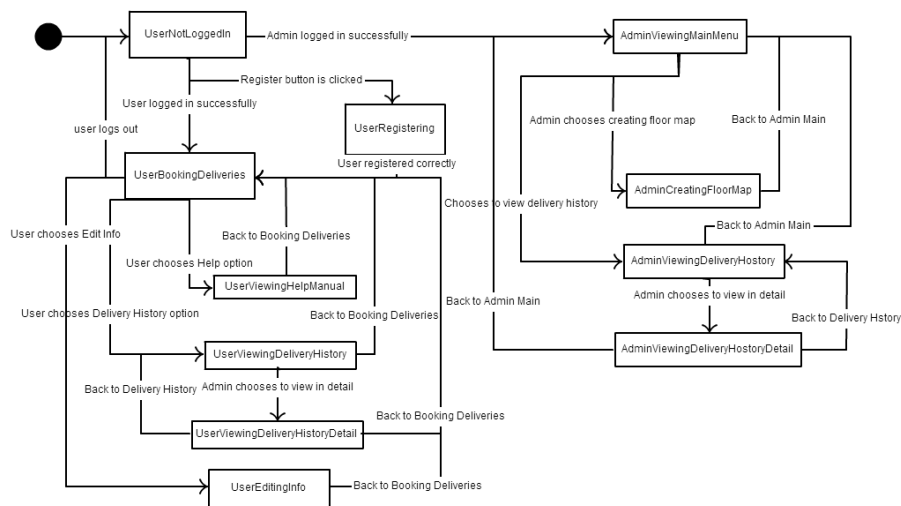


Figure 9.14.: State diagram representing the design of the transition between the view classes according to the user interaction with the system.

4. Motor Driver (reference [18]) which will be used to drive two motors of the robot for the sake of moving it properly.
5. 3 Reflective Optical Sensors (reference [16]) which will be used for the detection of the track (two parallel white lines drawn on the floor).
6. 2 Magnetic Sensors (reference [20]) which will be used for the detection of office locations in the working environment.

The most important requirement of this part is obviously a well-thought and well-designed software logic used for programming the arduino controller for manipulating all the hardware parts mentioned above. We, the developers, will guarantee that function normally.

10. ALGORITHMS AND DATA STRUCTURES

10.1. Algorithms

In this section, we will describe detailed algorithm of path resolution. Note that in Section 5.3 Mathematical Model in report 1, we described two math models used in our system, which are track detection and path resolution. As the first mathematical model has been detailed explained in report 1, with illustration and diagrams of state machines, we are not going into any further discussion about the first math model. What we will focus on is the critical algorithm that determines how the robot selects path to implement the work of picking up packages from the sender and delivering packages to the receiver.

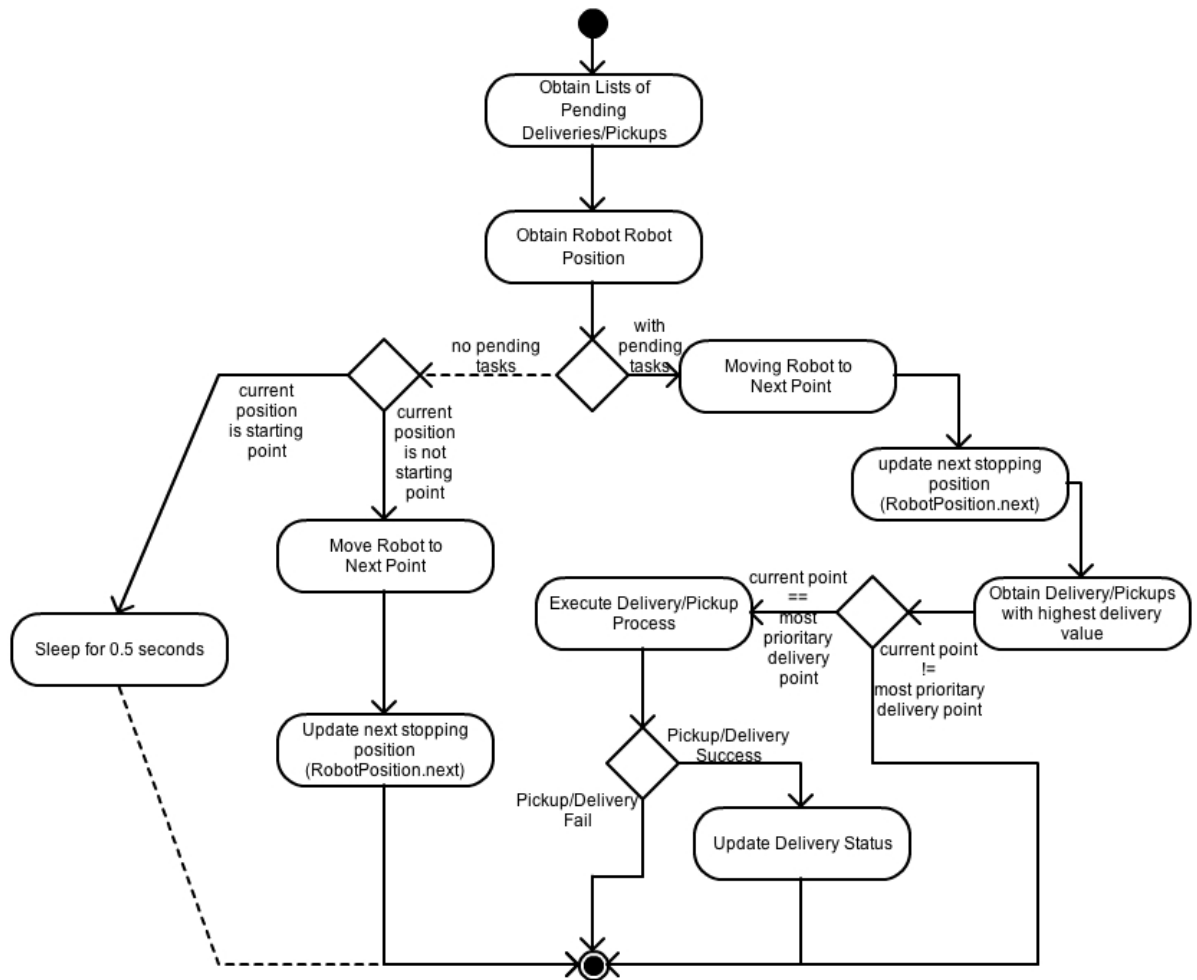
As stated in chapter 1 of report 2 and Section of report 1, there are several prerequisite for the algorithms to function normally:

1. The robot moves clockwise as long as it is moving
2. The corridor of the floor is a closed loop, where the robot can move to its starting point after move through the way all along the loop
3. The path that robot move through is covered with WIFI

The robot will report message to the server each time he arrives at a stopping point located by a magnet and detected by the magnetic sensor in the robot. Then the server will master the robot to start the algorithm of path resolution. This algorithm will be executed infinitely as long as the robot has the power to maintain his movement and the connection between robot and server is maintained. Figure 10.1 shows the activity diagram of how the algorithm is implemented. The starting point (black point in the diagram) of the algorithm in the activity diagram is each location point that the robot will stop, i.e the “real” starting point or each office that is shown as nodes in a floor map. The end point of the algorithm in the algorithm diagram is the end of the life cycle of this algorithm. So generally the algorithm describes the process in the time period between the moment of starting to move at current point and the moment of just arrive at next point.

Next we will illustrate each procedure of the algorithms step by step. The steps described below are in accordance with the activity diagram.

1. Step 1: the server will obtain a list of pending deliveries or pickups from the database
2. Step 2: the server will obtain position of the robot
3. Step 3: the server will judge whether there is pending tasks. If there is no pending deliveries(including pick-ups and deliveries of package), go to step 4; else go to step 8
4. Step 4: the server will check whether current point is starting point (HOME of the robot). If it is, go to step 1 if it's not, go to step 6
5. Step 5: the server will send messages to make the robot sleep for 0.5 seconds; then go to step 15
6. Step 6: The server will instruct the robot to move to the next point
7. Step 7: After the robot has moved to the position, the robot will update the next stopping position. Then go to step 15
8. Step 8: the server will instruct the robot to move to the next point
9. Step 9: the server & robot will update the next robot position that it is actually locate in currently
10. Step 10: the server will obtain most priority delivery, taking both the priority of the original value of the package and the position of the robot
11. Step 11: check whether current location equals to the location with the highest priority calculated in the former step. If the answer is yes, go to step 12;If the answer is no, go to step 15



[online diagramming & design] creately.com

Figure 10.1.: Checklist Chart

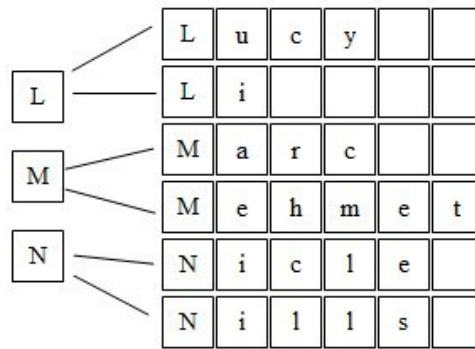


Figure 10.2.: 1-D reference arrays with typical words and 2-D array stored user names



Figure 10.3.: linked list with user accounts

12. Step 12: execute the delivery/pickup process. In this process the robot will ring the buzzer, wait for the sender/receiver to input his/her password and do the pickup/delivery
13. Step 13: check whether the delivery/pickup process has finished successfully. If yes, go to step 14; if no, go to step 15
14. Step 14: Update the delivery status into the system
15. Step 15: algorithm ends

10.2. Data structures

Data structures are necessary for any efficient software system. Our auto delivery system has all different types of complex data structures like arrays, linked lists, hash tables, stacks, and queues.

Our system uses arrays (see figure 10.2) because of their short processing time and arranging all items at equally spaced addresses in computer memory. Array is a data structure consisting of a collection of elements (values or variables), each identified by at least one array index or key. An array is stored so that the position of each element can be computed from its index tuple by a mathematical formula. In our system, we use two-dimensional array to store user account information and one-dimensional array as reference, and so user can search the information they need such as user name and mailing address by inputting some typical words. Since there should be numerous accessing to our system everyday from different users, we should concern about the performance of the system in terms of the responding time.

Linked lists (see figure 10.3) are other data structures that our system includes. This structure allows for efficient (the expected computing time is $O(1)$) removal or insertion of elements from any position in the sequence, and so provides dynamic use of memory. In our system, common user can create new accounts, add/edit personal information to the current account, delete the account if he/she wants; and the administrator can add/delete some records to/from the delivery history list. If we use linear list, these operations will be time-consuming (the expected computing time is $O(n \log n)$).

In computer science, a hash table (see figure 10.4) or hash map is a data structure that uses a hash function to map identifying values, known as keys, to associated values (e.g., mapping a name to a telephone number). Thus, a hash table implements an associative array. The hash function is used to transform the key into the index (the hash) of an array element (the slot or bucket) where the corresponding value is to be sought. In our system, we need to map keys of users' accounts to their personal information, such as password, first name, last name, address, and Email; and map some key numbers to delivery records.

Hash Address	Pointer
0	→ 33 Li Marc 11/08/2012 3 received
1	→ 34 Li Tony 11/12/2012 1 received → 12 Emma Tony 10/19/2012 5 received
2	→ 13 Mehmet Li 10/19/2012 4 received
3	→ 47 Marc Mehmet 12/06/2012 1 pending
4	
5	→ 38 Bowen Li 11/28/2012 2 received → 27 Bowen Emma 11/20/2012 2 received
6	
7	
8	→ 52 Tony Emma 12/09/2012 5 pending
9	
10	

Figure 10.4.: Hash table of delivery records (record index, sender, receiver, booking time, delivery priority, delivery status), Hash (key) = record index mod 11.

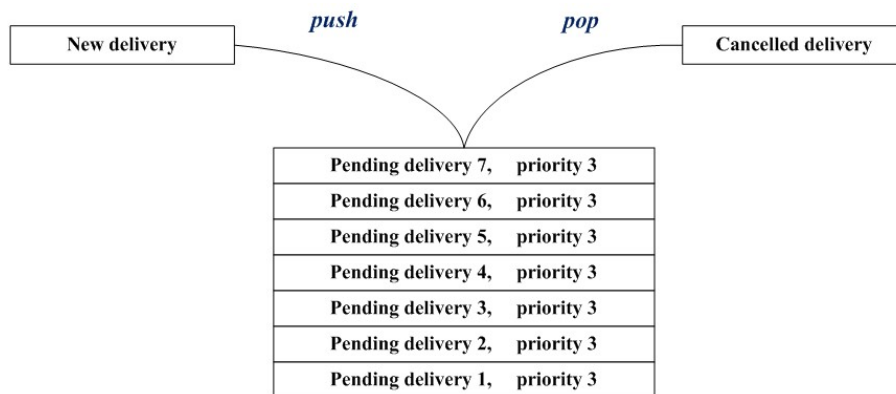


Figure 10.5.: Stack of pending delivery and push-pop operation

Stack (see figure 10.5) is an area of memory that holds all local variables and parameters used by any function. In our system, when there is free memory space, the stack will be created for operations need to storing information such as booking new delivery.

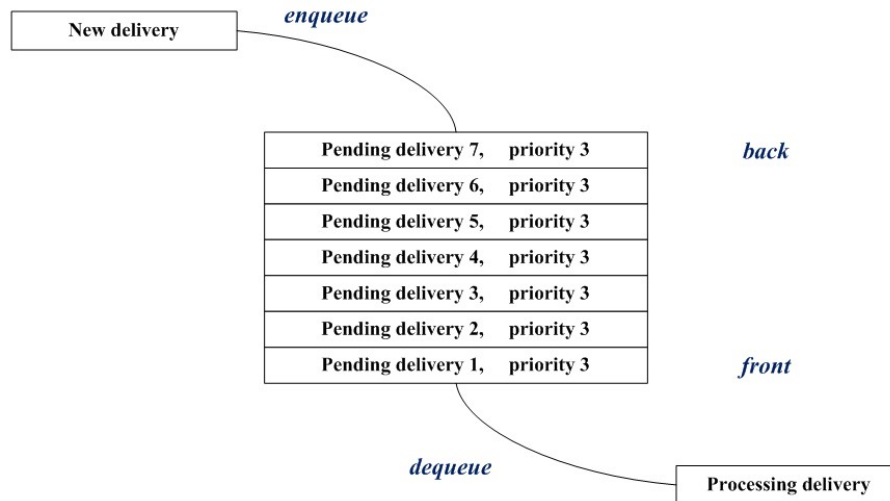


Figure 10.6.: Delivery queue and FIFO mechanism

As for queue (see figure ??), our system uses FIFO (first in first out) method. An example is that, when two deliveries have the same priority (the sender can set the priority from 1 to 5 when books a delivery), the delivery booked first will be processed first.

11. USER INTERFACE DESIGN & IMPLEMENTATION

11.1. User Interface Design & Effort Estimation

In this section we will explain step by step the graphical interface design of the most important use cases and will also estimate the user effort using the design.

Firstly, we will discuss about User Interface design of the use case Book Delivery. In section 4.7.1 we already talked about how to minimize the user effort in this concrete use case, but here we will repeat everything step by step:

- After the login, the booking screen will appear (booking is the main usage of the web/mobile interface) with a list of starred receivers and the 2 search boxes, as seen in figure 11.1.
- The user will click on the receiver, probably in the starred list, and the name will move to the first entry of an empty table (booking table), as seen in figure 11.2. Note that a *When to pick up?* question has appeared with a default value ASAP.
- To confirm the single delivery, the user only needs to click to the confirm button at the end of the page.
- To book multiple deliveries, instead of pressing the global confirm, the user will press another receiver from the starred list/search result table. The user will be allowed to repeat the process until the global confirm is pressed.
- Note that if the user searches something, the elements from the starred list will disappear, and in the same table the results will appear.

With this approach we reduced the experience to 3 clicks¹ and 1 screen. To book multiple deliveries, 1 extra click is needed if receiver is in the starred or 2 extra clicks if the user need to search.

Given the detailed flow of events shown in UC-2 Track Delivery Status, we can see that there are 6 different screens (Login, Check for 10 days, Check for n days, Search, Status display, Go Back) to accomplish this use case. Actually. For simplicity, we will reduce the user experience to the minimum and only focus on the several important situations. After login and selection of delivery status check, user can view the recent delivery information. If he or she wants to check one specific delivery, he or she can search for the result. After clicking the specific delivery, detailed information, including the package location and corresponding status at each specific time, can be tracked. See Figures

By clicking Help link, any person who is interested in the auto delivery system is able to visit help page of the system (figure 11.7). The help page consists of a search Bar and a Contents Bar. In the Contents Bar, the items include “Welcome”, “Index of Glossary”, and “Contact us”; when a user click the “Index of Glossary” item, the right side of the page will show several terms about details instructions about the system, which includes “Auto Delivery System”, “Book a Delivery”, “Register for an account”, “Send your package”, “Receive your package” and “Authentication”. A user can click any of the link to obtain instructions about the terms or operations in relation to the auto Delivery System.

11.2. Interaction of User Interfaces

From this section, we will show the actual implementation of User Interface. We use figure 11.13 to show the interactions between different user interfaces. Our software begins with a simple login window. As we

¹Login - Starred receiver - Confirm

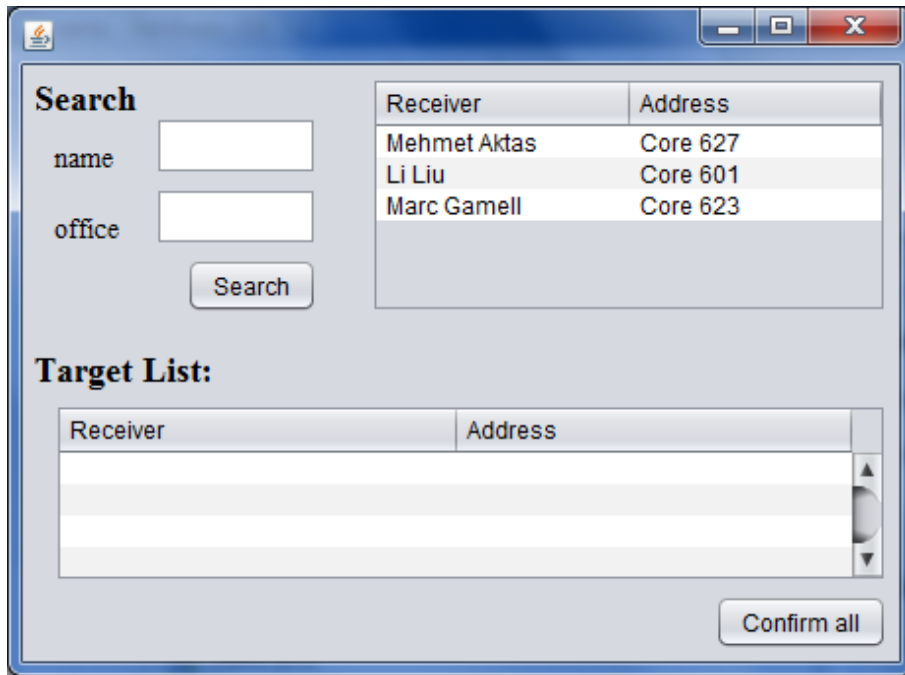


Figure 11.1.: First step of BookDelivery

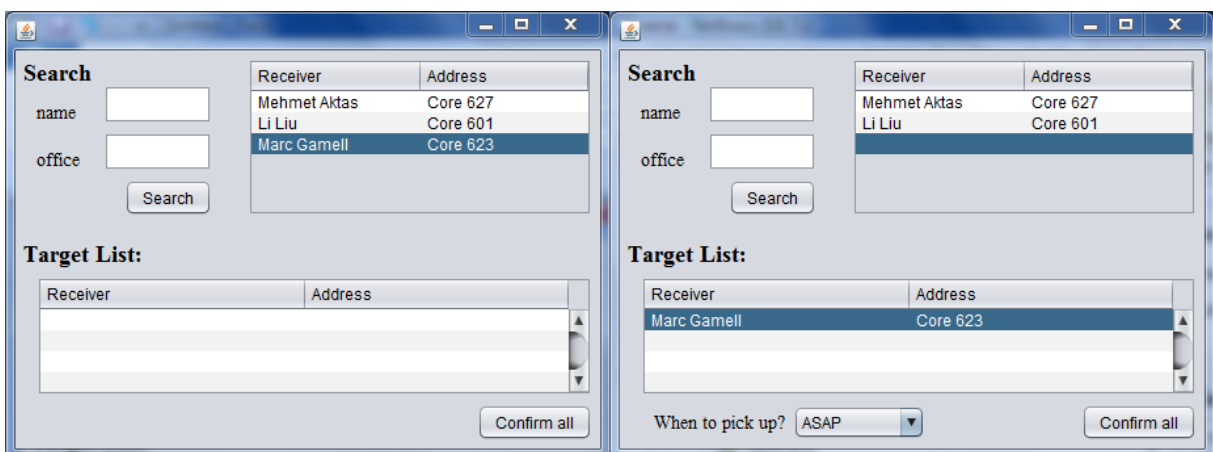


Figure 11.2.: Second step of BookDelivery

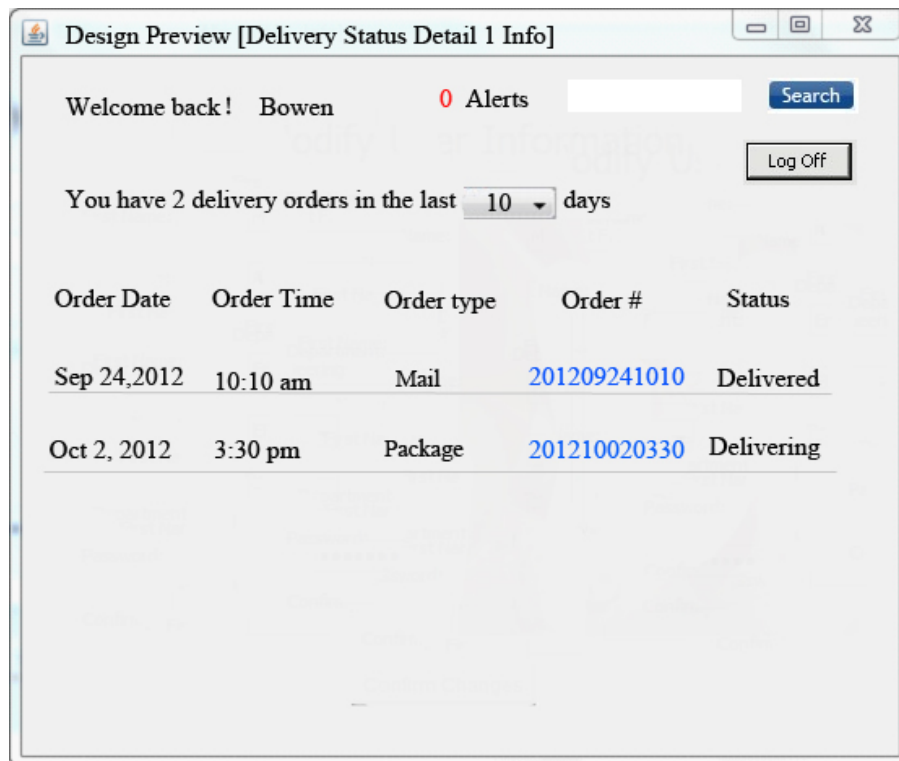


Figure 11.3.: Interface 1. A screen contains a list of deliveries (including time, date, sender, receiver, type and current status) within 10 days (10 is default, and it can be changed by user by using the selection box). A search box for user searching the specific delivery is shown on the right top.



Figure 11.4.: Interface 2. A screen of detailed information about one specific delivery, including the package location and corresponding status at each specific time. And this display is continuously updated.

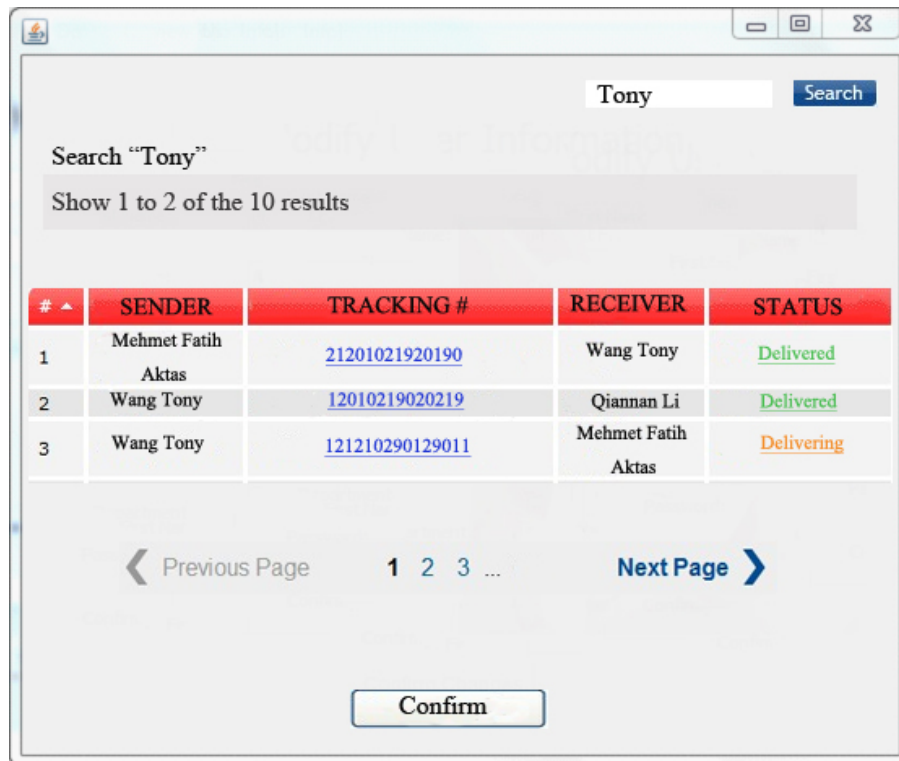


Figure 11.5.: Interface 3. After searching, a screen shows a list of delivery information associated with the searching word.

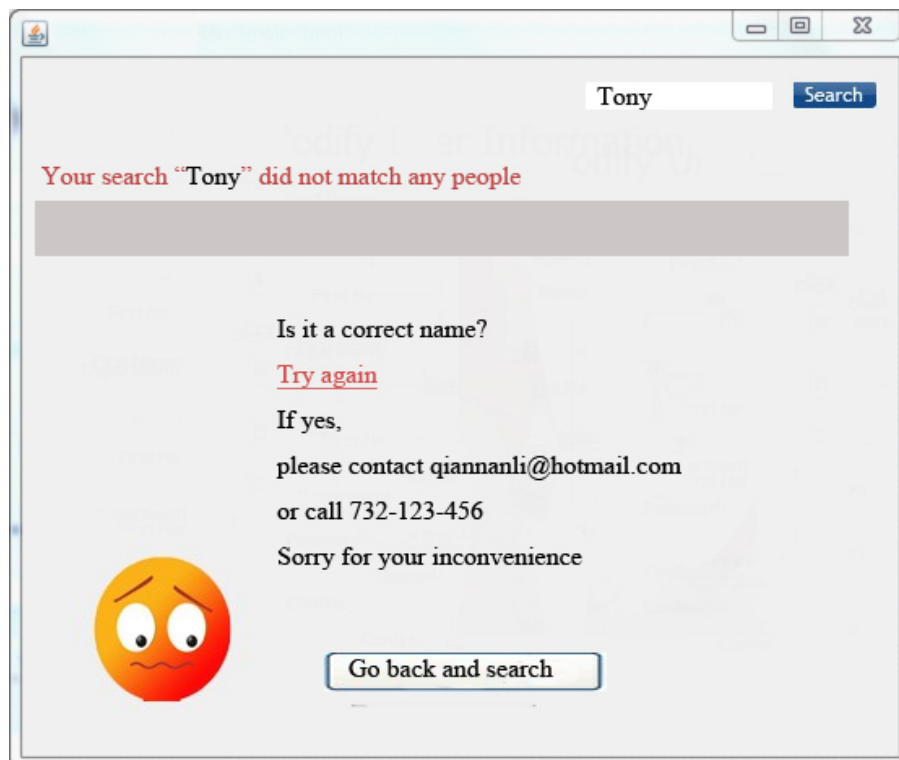


Figure 11.6.: Interface 4. After an invalid search, a screen shows that there is no match between the delivery and the searching word, a Go Back button is shown at the screen.

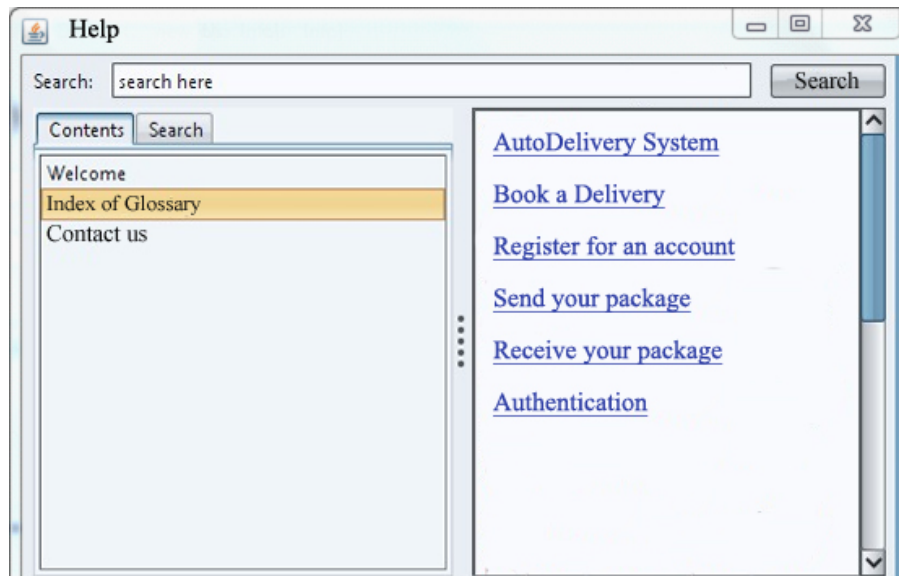


Figure 11.7.: Interface 1. Help page.

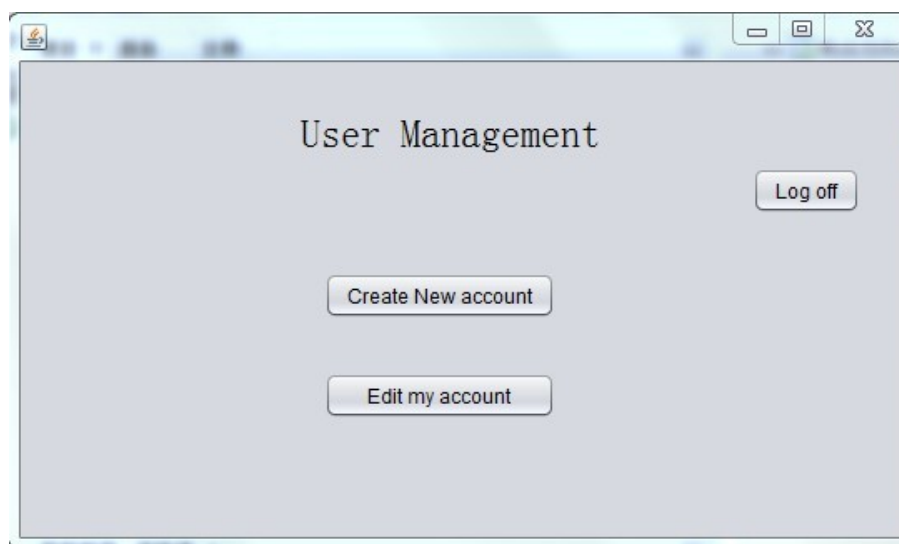


Figure 11.8.: ManageUser use case, interface for users



Figure 11.9.: ManageUser use case, interface for administrators

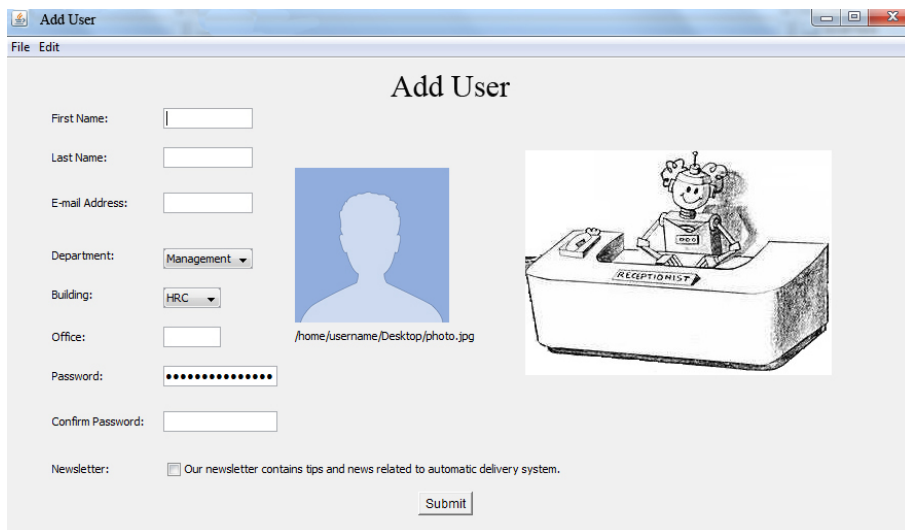


Figure 11.10.: Add user use case

Edit User Information

First Name:

Last Name:

Department:

Building:

Office:

Password:

Confirm Password:

Figure 11.11.: Edit user information use case

Remove User

First Name:

Last Name:

Department:

Building:

Office:

Password:

Confirm Password:

Figure 11.12.: Remove User use case

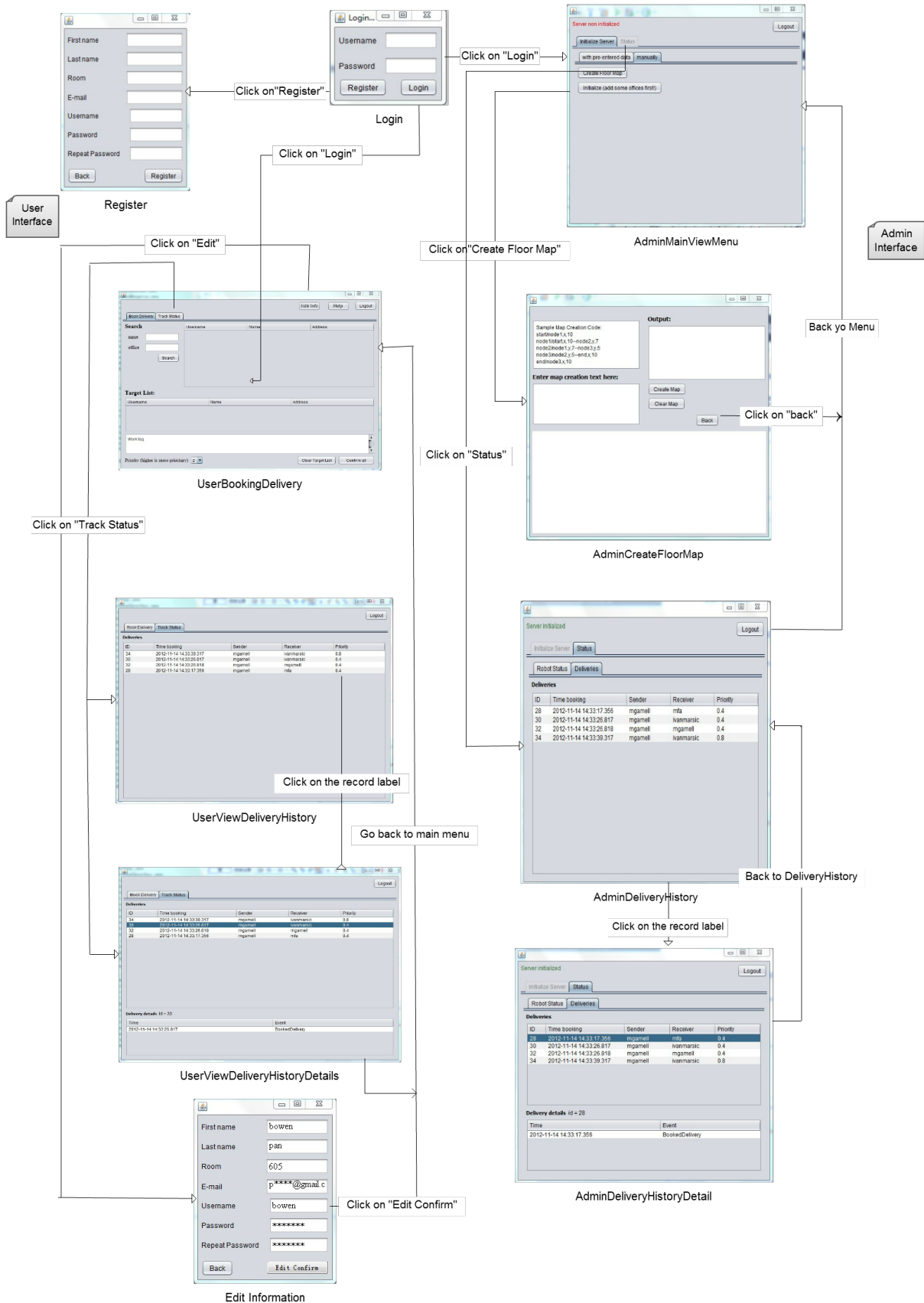


Figure 11.13.: Interaction diagram for the User Interface

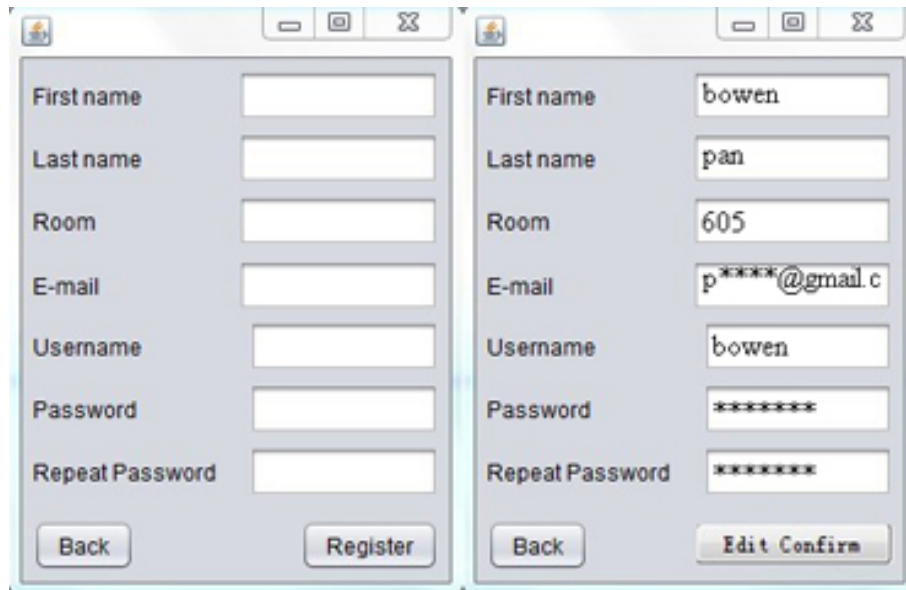


Figure 11.14.: Interface diagram for register and edit

can see, the user and administrator are prompted with different interfaces after login, so we will discuss the user's experience using this application separately, from the perspective of regular user and from the perspective of administrator.

- For a regular user, he or she is prompted with user main interface with two main options to choose from: Book Delivery and Track Status. When the user choose the tab 'Book Delivery', he/she can search the key words regarding the name or office number of recipient, then the search result will show up in an itemized pattern on the right box. The user can choose one or more (up to three) recipient into the Lower box 'Target List', and by clicking 'Confirm all' button, all deliveries in the 'Target List' will be booked into the system. The user can also choose another tab 'Track Status' in the main window, he/she can see a list of recent tracking history including items of Time booking, Sender, Receiver and Priority. The user can also see detailed status information of a specific delivery by double-clicking one delivery in the box. Detailed Tracking information will show detailed status of one delivery, such as one package may be in the state of 'BookedDelivery', which means the delivery has been booked but not been picked up.
- For the administrator with authorization, he or she will be prompted with administrator main interface with three options: create floor map, track robot position and view all deliveries with detail information. We define a simple language for administrator to create the plane graph of the office floor. For tracking robot position, we set up a dynamic picture monitoring the position and status of our delivery robot constantly. Also, the administrator has the priority to view the all the user's delivery records.

11.3. Detailed User Interface Implementation

In this section, we will discuss detailed user interface implementation, and we will show how users can get through the main functionalities of these usewith least effort and greatest convenience.

11.3.1. Register and Edit information

This is the user effort estimation of Register and Edit information, see figure 11.14

For register a new user

1. One Click on the register button

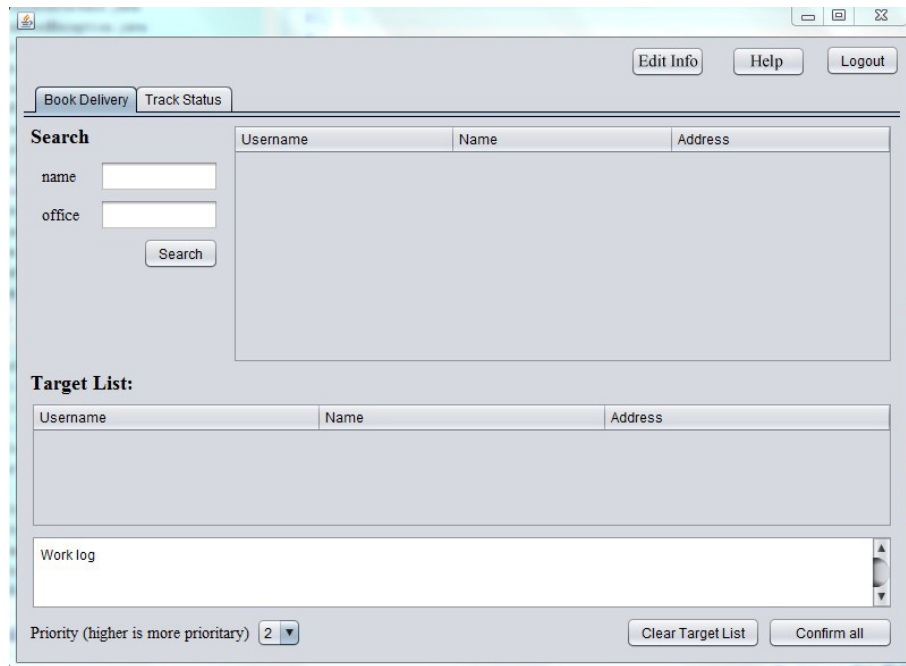


Figure 11.15.: Interface diagram for BookDelivery

2. Fill in the information by keyboard and One click on Confirm

For edit existing user information

1. Enter the username and password by keystrokes and one Click on the login button
2. Click on the Edit button on the top of UserMainMenu window
3. Fill in the information by keyboard and One click on EditConfirm button

11.3.2. Book Delivery

Next is the user effort estimation of Book Delivery, see figure 11.15 For users to realize one or more delivery booking

1. Enter the username and password by keystrokes and one click on the login button
2. Click on the BookDelivery tab button
3. Enter part or whole information about receiver, one click on confirm
4. Choose the target people by double click on the right and up side of target list
 - a) Double click on the target list in the middle textfield of the interface to cancel the chosen one
5. One click on the confirm all to finish booking delivery or deliveries

11.3.3. Track Deliveries Status

Next is the user effort estimation of TrackStatus, see figure 11.16 and 11.17

1. Enter the username and password by keystrokes and one click on the login button
2. Click on the TrackStatus tab button

There only need one more step to see the delivery detail

1. Double click on the record label to show the details in the bottom of the ver window

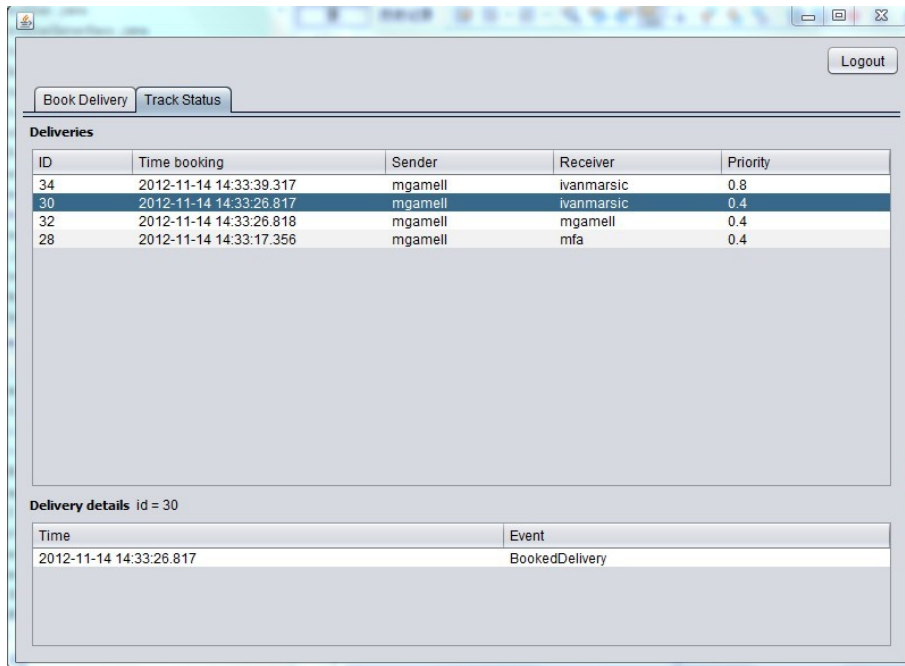


Figure 11.16.: Interface diagram for ViewDeliveryHistoryDetails

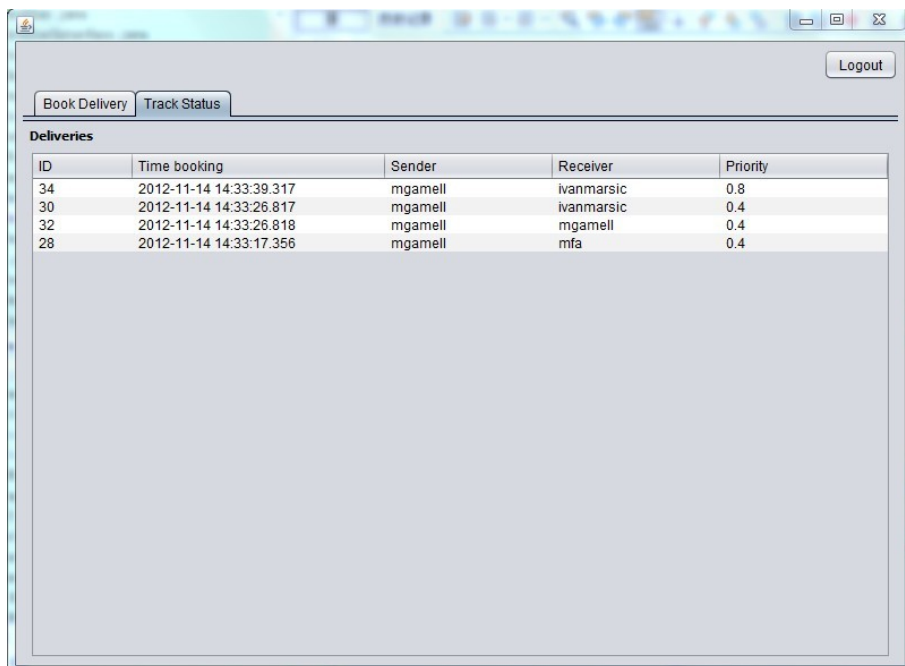


Figure 11.17.: Interface diagram for UserViewDeliveryHistory

Therefore, as we can see from the flowing procedure. To register, user need two mouse clicks with some information input. To edit, the user need 3 mouse clicks with some information input. To book delivery, user need 5 clicks for one delivery and $5+2n$ clicks for n deliveries. To track status, the user need 2 mouse clicks to show deliveryhistory and 1 more click to show detail. As a result, it basically satisfy our initial requirement demand.

11.4. Improvement in UI Implementation Compared with UI Design

We consider the user effort estimation when we program and design the actual interfaces. In our mock-up design, we depict a lot of interface. However, when it comes to the real interface design, we make changes to reduce user's mouse click and keystrokes to most extent.

- In the phase of user interface design, we didn't set up the priority options to distinguish different urgency degree and importance of mail deliveries. This may lead to big inconvenience for people who wants to send urgent package to someone or the package is of great importance and must be send prior to other packages. We add a drop down bar so that it's simple and fast for user to set up the priority value for his/her booking delivery. This priority number will be referenced when multiple packages are pending to be picked up or delivered, and package with greater priority will be delivered in a shorter time.
- In the mock-up design, we separate the BookDelivery and TrackStatus into two different windows. However, in interface design, we use the tab button to realize two main functionalities into one window. For users, there is only one or less mouse click to reach either BookDelivery window or TrackStatus window.
- For the BookDelivery part, we simplify the user effort of keystrokes by providing function through searching by name or by office number. For example, if we want to send a package to Marc Gamell in office 607, we don't need to enter the whole name and office number. Users can type a m or mga then click on confirm button, then choose the correct one by one mouse click from the possible answers shown at the right text field.
- In the mock-up design, we initially aim to create an interface that specifies the delivery details. However, as the figure 11.13 shows, we incorporate this table into the bottom of deliveryhistoryview interface. Users can easily check the detail information only to one-mouse click on the record label, then the detail information will come out in the bottom textfield area.

12. DESIGN OF TESTS

12.1. Test cases in Unit Testing

12.1.1. Strategy to Do Unit Testing

In this section, we will discuss about the strategy we are going to use in unit testing, the method on how to choose test cases and the description of test cases.

In Section 2.3 Acceptance tests for User Stories, we have designed several acceptance tests for each user stories. Each of them represents a coarse way of testing some specific user-stories. For example, user story No. 3(ST-3) says "As a sender, I can send a package to the receiver based only on his or her name, so that I don't need to care about his or her location". The corresponding acceptance test cases in report 1 include testing whether name and office information will be requested in the registration period, whether requested name will appear on the search result, and whether name and office information could be modified by each user in the system.

Similarly, in Section 3.8 from Report 1, we draw several tables to test cases for some important use cases. For example, Test Cases for UC-1: BookDelivery illustrate in a very detailed manner on how the Use case BookDelivery could be tested, by dividing this into six scenarios. We may want to implement our unit test by using existing test cases illustrated in report 1 as previous stated. However, current developing progress and our system characteristics also need to be taken into consideration: As already detailed illustrated in Chapter 3 of this report, our system is composed of three main parts: Central Station (know as the "server"), user computer, and the robot. Meanwhile, we use server/client architecture for connection between central station and client-computer, and we use master/slave architecture for connection between central station and the robot, testing each test cases in acceptance test of user stories or from test cases of user-stories may seem to have too much coupling among the server, client and the robot. Another reason is that currently the hardware materials to assemble the robot are not ready yet, thus no detailed implementation will be done in the robot side before demo1. Gathering all these information, we decided to choose a more detailed way to do the unit test compared to the two methods mentioned before. Since we are writing codes using Java IDE NetBeans, we choose to use the JUnit tool installed in this IDE to do the unit testing. We will write test codes in accordance with main packages and classes in the functional codes.

12.1.2. Test Cases in Unit Test

JUnit test fixture is a Java object, and we can use the @Test annotation to write each testing method for the class we would like to test (reference [13]). We plan to write test code just in this way to test public methods in classes we defined in the code (the code that contributes to the functions of the system). The test cases that we are going to implement are categorized in this way:

1. Test Cases in package "presentation": As functional codes in this package can be differentiated into two parts: client-side control logic and drawings of user-interface, we will handle differently according to these two types of classes.

In class ClientController, which handles with all state changing regarding showing different user-interface windows, we will run the method that initiate some certain state (such as entering the LOG.IN state), and see if the state indicator has changed to the corresponding state.

In classes about showing interface to the user, we would like to test in the way that initiate the constructor of the user-interface class, and see whether the interface will actually show up.

2. Test Cases in package "logic": As this package handles with the main logic in server-side, such as sending indication to manage user account, getting delivery information that has the highest priority, we are going to test whether these logics are correct. For example, we would like to pre-entertain some delivery information and robot position, and test whether the logic about getting most priority delivery can be executed correctly. We are also going to test using pre-defined database information as stubs, to detect whether the server-side can communicate correctly with database.
3. Test Cases in package "resources.controller": Functional codes in this package handle with communication interface between server and robot. Since the real communication protocol has not been set up yet, we are going to test this part by detecting whether communication interface can correctly receive information from the server.
4. Test Cases in package "resources.data": Codes in this package define some low-level data structures and basic transitions and interface to higher level. Our test will do the following thing: define some data structure, and test whether the interface and transition works! (such as testing whether the toString of a specific data type works)
5. Test Cases in package "resources.datacontroller": As this package handles with the set-up and interaction to the database and some direct control to the data passed from upper level (such as check whether username and password is correct), we will test codes in this package by checking whether these controls actually work (such as we update the position of robot and see whether it is actually updated from visiting database).

12.2. Test Coverage

Code Coverage can describe the degree to which the source code of a program has been tested (reference [9]). It can create a loop of revising the code by doing such things (reference [15]):

1. measuring code coverage
2. write tests
3. run tests
4. fix bugs
5. repeat the loop

We used the EMMA Code Coverage plugin tool to do the code coverage.

12.3. Integration Test strategies

With regard to the integration test, we are going to choose the "big bang" integration approach. Specifically, we will do the integration test using the "Bottom Up" Integration approach. We choose this method because our system (here we refer to the server/client-side), has the 3-tier architecture. Therefore, it would be more convenient if we test from the bottom layer's resource layer, to the upper layers, which is the logic layer, until at last, the presentation layer.

The classes in resource layer, which is the bottom layer of our architecture, should be tested first. Each class in this package should have already been tested as part of unit test.

Then we can do integration test from the upper level, which means doing testing from the perspective of logic layer. In this layer, we will do three integration tests, as states below:

1. Delivery Coordinator integration test: In this integration test, we will try to write a driver to drive delivery coordinator to coordinate with several deliveries. The test stub will be the whole lower layer of resources.
2. System Status integration test: In the system status integration test, we will try to write a driver to drive the System Status checker to check information about robot position, status of robot moving, and the user's position. The stub will be the whole lower layer of resources.

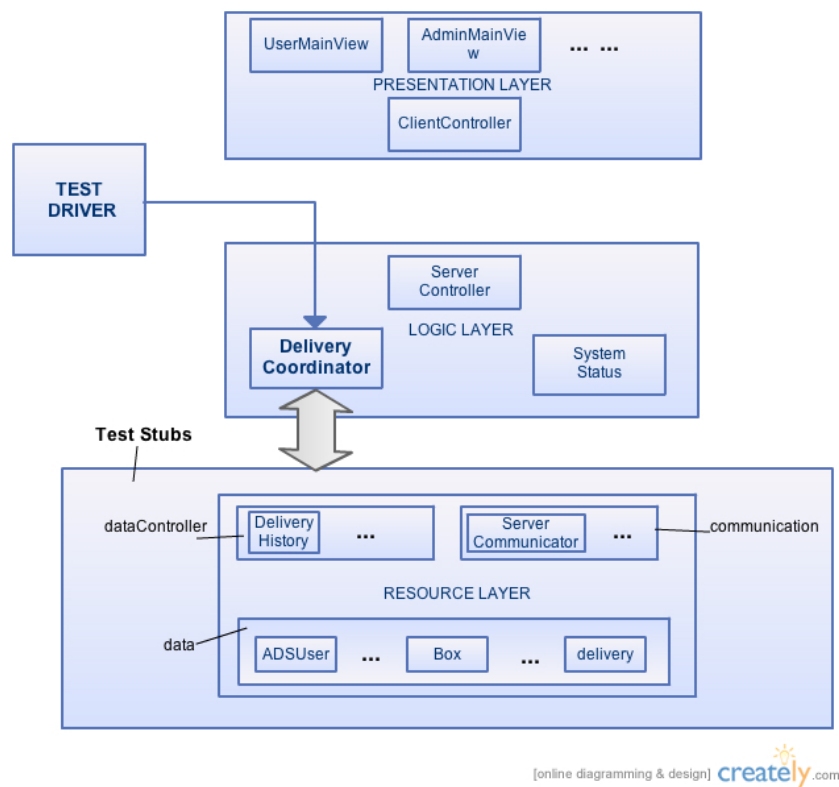


Figure 12.1.: IntegrationTest_Flowchart

3. Server Controller integration test: In the Server Controller integration test, we will put even delivery coordinator and system status checker into the integration stub as well as with resources layer. We will write driver to drive system controller to do bookdelivery, delivery/pickup, and system status view together as a full integration test of server side.

In the upper-most layers' presentation layer of the system, the integration test will be done as below: all two lower-level layers' logic layer and resources layer will be treated as a whole stub. we will write drivers to drive each view (each window that interacts directly with the user), and testing inter-connection with the stub.

Figure 12.1 shows an example of doing integration test with Delivery Coordinator. As we can see in the picture, Delivery Coordinator and its lower-level resources layer is treated as an integral entity to be tested. Test drivers activates the delivery coordinator, and the delivery coordinator use the test-stub' the whole resource layer to check the correctness of delivery coordination.

13. HISTORY OF WORK, CURRENT STATUS, AND FUTURE WORK

13.1. History of Work

In this section, we are going to discuss the history of our work. We will review how we have accomplished each milestone in our project.

- **September.7-September.22:** At the beginning of this semester, we discussed and set up the general topic of our project: an automatic system aiming at delivering packages in the office building. Then we start writing proposal of our project. We include our solutions regarding office delivery and claims general user experience we will supply to the user. On September. 22, we finished and submitted proposal.
- **September.23-October.3:** During this time, our group was focusing on finishing part 1 of report 1. We were refining customer statement requirement from our proposal, gathering and organizing user stories we would like our system to have. We have specified the features of our system from a general way into more specific functional and non-functional requirement by discussing what kind of user stories we would have.
- **October.4-October.10:** Our group was focusing on finishing part 2 of report 1. We expanded our functional and non-functional user-stories into 19 use cases. Then we draw use case diagrams and test cases according to each use case. Also, we draw system sequence diagrams which contains interaction between actors and system, such as how a user can get involved in picking up package. What's more, we also designed our user interface, such as window of log-in, booking delivery, and viewing delivery status.
- **October.11-October.14:** Our group was mainly doing design analysis of this system. We draw the domain model, which contains domain concepts in the server part and concepts in the robot part. We are also analysing mathematical model which would be used in our system, such as algorithm to make the robot follow the track, and the algorithm to maintain a picking up/delivering package order. On October. 14, we submitted our report 1, which means we have basically finished design part of the system from a higher level.
- **October.15-Nov.9** Our group was doing paper work and programming in parallel at this time. We drew interaction diagrams for several important use cases, such as BookDelivery and PickUp-Package, by using our domain concepts mentioned in report 1. We figured out what the concrete relationship between important domain concepts. We also finish the class diagram and interface specification, which contains detailed class description in our system. Another paper work we were working at is the system architecture, which contains all architectures, data storage methods, network connection methods we are using in our system. At the same time, we were doing programming in the server side and client side. We were using the 3-tier architecture and Server/Slave architecture to program the client, server, and resources (which are three main packages in the server/client side). On Nov. 8 we submitted our part 2 of report 2. On Nov.9 we did our demo No.1 in the class. In the demo, we showed full feature in the server side and client side. A user is able to register, log-in, book a delivery, and view delivery status by using the system. We also created a robot stub (which is a graphical stub as a window) in the demo to show what the robot will behave in our system. We left the actual implementation of robot part to the demo No. 2.
- **Nov.9-Nov.15** Our group was doing remaining part of report 2, including algorithm and datastructure, user interface implementaion analysis and design of tests. We illustrated how our mathematical model in report 1 has evolved into this detailed algorithm of managing the process of picking up packages or delivering packages. We were also discussing the improvements we had made of our

actual user interface compared with original design and mock-ups. On Nov. 15, we submitted report 2 of the project, which matches the our goal and requirements of the class.

- **Nov.16-Dec.12** We were focusing on the implementation of the robot. We implement the whole robot step by step. In the first step, we were doing hardware connection and basic debugging to the Arduino board. We were trying to make the motors connected with the wheels of the robot being about to run and stop; In the second step, we were trying to make the robot follow the line, which is the black tape we set up as a tray for the robot to follow. We set up six optical sensors and programming the follow-the-line algorithms referencing the interaction diagrams and algorithms analysis in report 2; In the third step, we set up magnet sensors, and LEDs as open/close sign of each section in the robot; In the last step, we were trying to make the robot move smoothly and being about to connect with the server side. On Dec. 12, we were doing demo 2 of our system. We presented our system with almost-full features implemented to the class. From the server side, our system is able show an interface with users to handle with their delivery requests and manage user information. From the robot side, our system is about to interact with user by moving the robot to the office of the sender to pick up the package, and then moving to the receiver to delivery the package. Our system also has the feature of multiple delivering, which means that multiple delivery requests can be handles at the same time. We were also doing merging of report 1 and report 2, and revise each chapter according to the requirement and reviews from other groups.
- **Dec.13-Dec.15** Our group was doing remaining work of report 3. We were doing effort estimation of this project to evaluate duration and velocity of this project. We were also doing design pattern description and OCL(Object Constraint Language) description of report 3. On Dec. 15, we submitted report 3, which matches our goal and requirement of the class.

13.2. Current Status

In this section, we are going to talk about current status of our project after demo 2. We will discuss about the communication mechanism among software components of our current Auto Delivery System. Then, we will illustrate what kind of features we have implemented and shown in the demo 2. Especially, we will talk about the movement and function of the robot part, because it is the main component we are focusing on during the second-half of the semester. Further more, as the robot itself is an embedded system containg a lot of hardware and other non-software techniques, we will discuss the main non-software problems we faced when developing the robot and how we fixed this problems and bugs.

13.2.1. Communication Mechanism among software components

As we can see from figure 13.1, our Auto Delivery System contains the following software components(from the higher level): Java Server, MySQL database, Java client, PHP+AJAX+jQuery client, JEE wrapper, Arduino robot, Motor and Speaker.

In Demo 1, three components of our system can running normally, which are Java Server, MySQL, and Java client. The communication mechanism between MySQL and JavaServer are JPA (Java Persistent API). We use JPA to store, update, and fetch information to MySQL database. We use Java RMI to communicate between Java Server and Java Client. By using Java RMI, the in-fact remote objects can be treated as local ones, and it also support multi-threading.

Please note that the other parts of the software components are all finished during the time between first demo and second demo. We add another kind of website client - PHP+AJAX+jQuery client. In order to make this client able to communicate with Java server, we need a JEE wrapper to encapsulate data in the web client so that it can send and receive message using Java RMI with the Java Server. Note that the PHP+AJAX+jQuery client use CGI (Comman Gateway Interface) to pass data back and forth to and from the JEE wrapper.

What's more, we also developed the robot with Arduino. The Java Server communicates with the Arduino WiFi shield using TCP protocol. We create a TCP package each time when a server need to send a message to the robot, and each time when a robot need to return information to the server. This

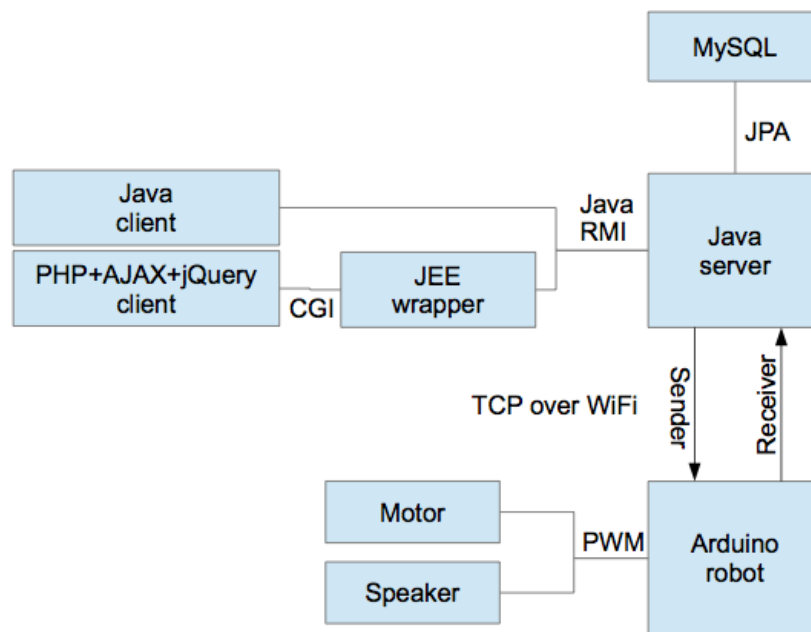


Figure 13.1.: Communication among Software Components in ADS

is not fast but can reduce the definite timeout when using socket communication between Java Server and Arduino WiFi shield. The Arduino board use PWM output to control the Motor and Speaker.

13.2.2. Features in Auto Delivery System

Currently, our system has below implemented features, which contains almost all features in the use cases.

From the **server/client side**, our system has the following implemented features:

- The system can show up a log-in window to the user from the user's computer, or visit the website. The user can choose to log-in, or register, by using this window.
- The system can provide searching function for the user to search for receivers using a few words of his/her name or office location.
- The system can provide "Book Delivery" function to the user, when the user choose one or more names in the search result pane, and click 'Confirm'. Note that Deliveries that are booked can be more than one at the same time.
- The system can provide the user delivery history information if he/she clicks the 'Track Status' tab in the main menu.
- The system can provide the administrator delivery history information of all users in the system.
- The administrator can draw a floor map before the system is put into work in a designated floor of the office building.

From the **server/robot side**, we are going to first explain what our robot and the map is like, then we will discuss the features of the robot one by one.

As we can see from 13.2, Below we listed several components in the robot, which are:

- **A: Motors:** There are two motors in our robot. They are in charge of doing the actual 'moving' and 'stopping' operations driven by the Arduino Motor Driver. For example, if we want the robot to move forward, then we set the two motors running at the same speed. If we want the robot to turn left, then we stop the left motor, or let it run in a very low speed, while driving the right motor running at a higher speed. In this way, the robot is capable of turning left. The method of making the robot turn right is similar. Please note that we simplify the original four-wheel design

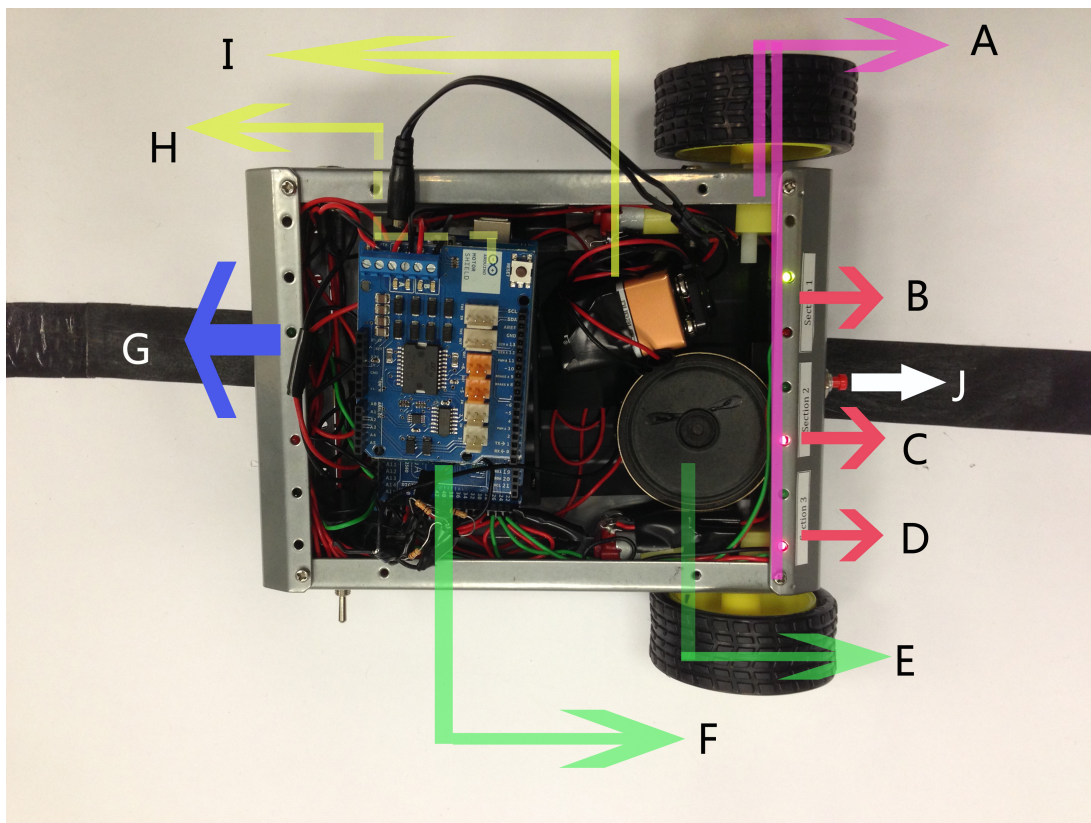


Figure 13.2.: Components in robot

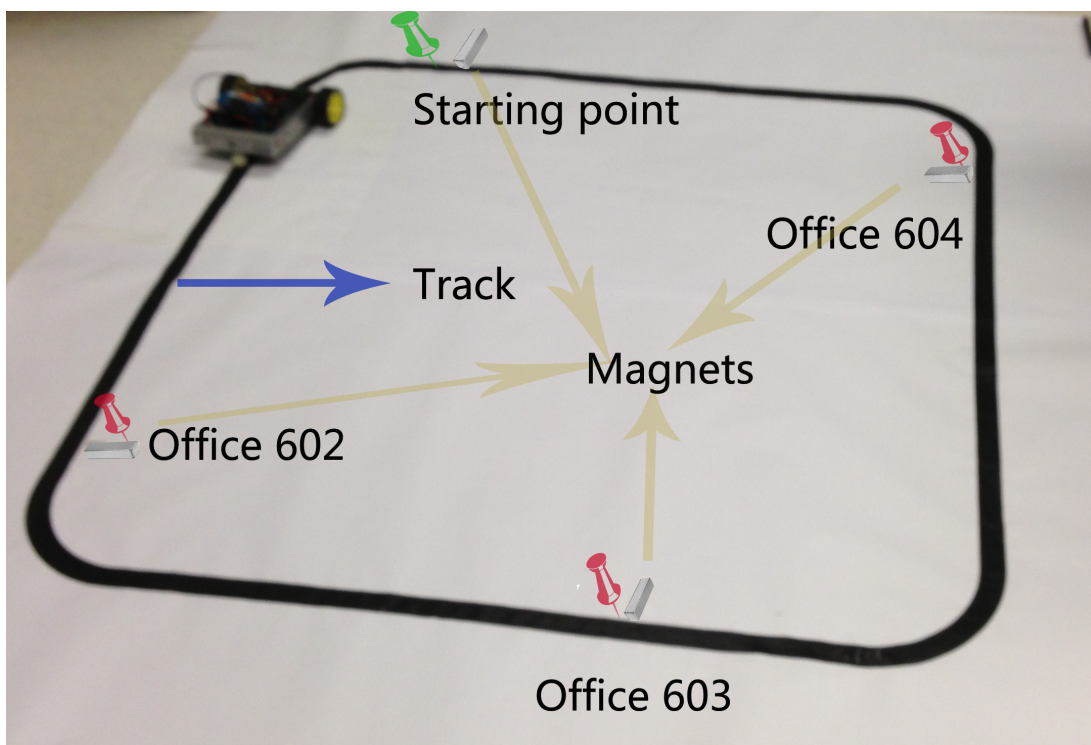


Figure 13.3.: Floor Map in demo 2

to only two-wheel design, in this way, the turning and ‘following the line’ will be much easier to control and achieve.

- **B: Section Open/Closed Indicators 1:** As we can understand from meaning of the sticker ‘Section 1’, the two LEDs closed to the ‘Section 1’ are indicators for the open and close status of this section. When the ‘green LED’ is turned on by the controller, it means that Section 1 is open, and users are available to put package into this ‘Section’ or take package out from this ‘Section’ (Currently there are no real sections with a ‘cover’ and a ‘lock’ in our robot, and we only show the control logic for each section). When the ‘red LED’ is on, it means this section is closed.
- **C: Section Open/Closed Indicators 2:** This part indicates the control logic of Section 2, with a red LED and a green LED next to the sticker ‘Section 2’. Detailed description is similar as **Section Open/Closed Indicators 1**.
- **D: Section Open/Closed Indicator 3:** This part shows the control logic of Section 3, with a red LED and a green LED next to the sticker ‘Section 3’. Detailed description is similar as **Section Open/Closed Indicators 1**.
- **E: Speaker (Buzzer):** This part is one of the output to the user of this system. When the robot arrives at an office and want to pick up a delivery or deliver some package to the person in this office, the speaker will ‘buzz’ once and wait for the person to come.
- **F: Arduino boards:** The Arduino boards are the core parts of robot. Three boards, which are ARduino Mega 2560, WiFi shield and Motor shield, are connecting together. Arduino Mega 2560 is the controller. WiFi shield is in charge of communicating with the server, and motor shield is used for controlling two motors we have.
- **G: Optical Sensors and Magnet sensors:** In the back of the robot, there are totally 6 pairs of optical sensors and white leds, and 2 magnet sensors. We use optical sensors and leds to detect the track (which is a circle of black tapes on a white paper, we will discuss this later). We use magnet sensors, which is technically magnet switches, to detect the magnet located near each office. When the magnet switches sense a magnet, the system will know the robot arrive at a new office.
- **H: Power Supplier for motor shield:** This power supplier is used to give power to the motor shield.
- **I: Power Supplier for Arduino Mega 2560:** This power supplier is used to give power to the Arduino Mega 2560 controller.
- **J: Button:** In our system this button have multiple functions. It is used to ‘simulate’ as password checking process, because the keyboard we bought cannot work properly. This button is also used as ‘Confirmation’ button when the sender has put his/her package into one section of the robot or when a receiver has take his/her package from one section of the robot.

As we can see from the figure 13.3, the offices located on this floor are office 602, office 603 and office 604. There is also a ‘Starting Point’ where the robot will start from and stop at when there is no deliveries pending. We use black line tapped onto the white paper to simulate the ‘track’. The robot will move exactly along the track because of the white LEDs and optical sensors located on the bottom of the robot. There is a magnet in each of the office, as well as the ‘starting point’. Magnet sensors located on the side of the robot will detect these magnets and stops at the office if needed.

Now we will begin discuss about the features implemented in the server/robot side.

- The robot moves according to the circular black line which was set up first as the track of the robot. The robot will stop for a short period of time whenever the magnet sensor detect the magnet nearby. It will send message to the server and update location information in the server side.
- The robot will stop and ring buzzer at the office where it needs to pick up the package from this office or delivery the package to this office. If the sender or receiver didn’t appear and press the ‘button’ to authenticate password until timeout, the robot will leave and move to the next point.
- The server will send an email to the receiver when a delivery is booked by the sender.
- The robot will request password authentication before the sender can put package into the robot or before the receiver can get package from the robot. Password authentication is finished from the server side by communication between the robot and the server. What now the sender or receiver

need to do is to press the 'button' as a sign of password checking process. We assume authentication will always pass because we need further steps for pickup/delivery, while currently the keypad is not available to use. What we do now is to send the user's correct password to the server as if he/she has input them. Then the server will authenticate and send "open the section" instruction to the robot. The designated section that is going to be used by the sender or the section containing the receiver's package will 'open'. Because there is no real section in the robot, what we will see is the green LED of corresponding section will be on after the server send message to the robot.

- The robot will request the sender/receiver to press the button again when he/she has put package into the section or take package from the section. Pressing this button is regarded as a confirmation that the interaction between the robot and this person is over at this time, and the robot is available to leave or execute the next Pickup/Delivery.
- The robot will go to the package Pickup/Delivery with highest priority when multiple deliveries are pending. This is to say, when new deliveries are booked, the robot will recalculate the booking/delivering priority when it arrives at the next available office location. As the communication between the robot and server are now quite low (using TCP package), the robot will wait at each office location for the response from the server about what the next pickup/delivery office location would be.
- The robot is able to handle with multiple deliveries and even new booked deliveries was added during the time the robot is executing other delivery work. For example, when the robot finished delivery to the person in Office 603, this person at the same time booked a delivery using his pc. Then the robot will stay at this point and request a new password authentication to this person. He will need to press the button again and when the green LED of a designated section is on he/she put the package into the section and press the button again for confirmation.
- The robot reports updates to the server so that delivery status updates will be shown on the main menu when the user request it.

13.2.3. Non-Software Problems and Solutions

In this section, we are going to discuss the non-software problems we were facing and how we solve these problems.

- **Problem and Solution to Motor shield not working continuously**

P: We could not make the motors run continuously. They were cutting off regularly, no matter how we change the software-controlled speed of the motor.

S: we thought the problem is with the motor shield itself, because the motor shield is not made from Arduino, but some other brands claiming to be compatible with the Arduino board. Then we bought another motor shield, which is original made from Arduino, and saw that the problem still persists with the new motor shield. Then we are trying hard to find other possible reasons to this problem. One interesting thing is that, when Mehmet, one of our group members, accidentally touch the motor shield while it is running, it seems that the 'cutting off' problem disappeared. Finally we figured out that this is because when touching the motor, the body acts as a capacitor and it can reduce the noise brought about in the robot system. Thus, after several trials of capacitor soldering between the tabs of the motor (from small capacitance values to larger). Then Right motor begin working seamlessly once we connected one 100pF and 10pF parallel (110pF totally). And the left motor become perfectly running by just one 10pF capacitor might.

The problem is due the high frequency noise coming from the power supply and the whole system of motors and capacitors begin to act like a simple RLC high frequency filter whose cutoff frequency is found by us by trial and error method.

- **Problem and Solution to Wireless connection between robot and server**

P: The TCP connection (between the robot and the server) reset happens randomly. This problem is also encountered during demo. As a bug of Arduino Wifi shield, a TCP connection cannot last more than around 13secs. We see this phenomena from the log printing of terminal.

S: To try to solve this issue, creating one TCP connection for every send/receive session is used. A cost of this is to introduce additional delays because synchronizing packets between the server and the robot take a lot of time. Also, the Wifi shield of arduino is not really fast about transmitting and receiving! This approach is a reasonable way to solve this problem in most of cases, while we admit it is not that efficient since the data transmitted between the robot and the server at one packet was really small. This is to say, we are using long connection sync overhead for small amount of data. However, it's an available way of solving this problem.

- **Problem and Solution to Wifi shield library**

P: Wifi shield library has also a critical bug. By using this library one cannot create and keep two connections alive. The reason that we tried to use two connections at a time is to find another effective way of solving the previous problem. Long delay introduced by sending TCP packet one by one is a critical issue for making the robot react fast and be 'smart' enough.

S: We tried to make server to send a message asking for response from the robot side, to keep the connection between the server and the robot. The message is just like saying "Are you still active?", and the messages is sent from the server for every 5sec to make the TCP connection alive. If at least one pile of data is transmitted over the created connection within each possible 13secs interval, then the problem of automatic connection reset seemed to be resolved. However, this approach introduced another problem. If robot sends a message indicating "I am alive!" and another message "I reached a node!" message consecutively within a small time interval, then server may possibly get wrong message corresponded to to this message because this is not exactly what the server expect to receive. To solve this we tried to open two TCP tunnels (sockets) with different ports and transmit these two kinds of messages over different tunnels. But it did not work because of this freaking bug of Arduino Wifi shield (Not allowing more than one connection at a time). That is why we keep using the solution of previous problems, which is, to create an TCP connection for every send/receive session.

- **Problem and Solution to the debugging messages causing loop running slow**

P: Debugging messages in arduino loop slows a lot of the loop. We were using serial debugging between the robot and the server for sending some debugging messages from the robot to the server. But since this communication is done serially it was really slow. And also in arduino side program, there is a function called "loop()" which can be thought as a main function with a difference that it is running forever, as long as there is power supply. When we used these debugging messages in the loop() this made the loop() to be run not as frequently as it should be. That is why robot begins to not working as expected. The operations begin not to be done very frequently but instead, very slowly. That is why robot begins to work inconsistently and react incorrect to the optical sensors and have trouble following the track.

S: Simply try not to use so many serial debug output.

- **Problem and Solution to the 4-wheels cannot working**

P: We chose a 4 wheeled large case for our robot for the sake of being consistent with the expected result of our goal of making this project. That is to say, our robot need to be strong enough to carry multiple packages at the same time. Using 2 wheeled small circular robot might not be very convincing in terms of providing a strong robot to carry packages, but we could not achieve to turn effectively with the 4 wheels because of mechanical reasons.

S: We at first tried to find a small caster to use at the front side of the robot instead of having two wheels and asked the mechanical engineering people to get one of those but the ones that they came up were really big for our robot. Then we decided to use a teflon ball (at the front side), instead of two front wheels, which can slide easily on the floor map. Thanks to the guys in mechanical engineering department.

- **Problem and Solution to the minimum motor speed**

P: The motors that we got started working at a very large speeds, even we have change it to the minimum speed. That is why, we could not make the robot move with slower speeds. The possible reason of this, is again wheels road grab power, motor power, weight of the robot, the floor that the robot is moving, which are in the field of mechanical engineering.

S: Try to optimize the movement of the robot on the floor by making it to go at high speeds, instead of trying to lower its speed.

- **Problem and Solution of having difficulties to tune the PID parameters**

P: Since the robot was moving at large speeds as stated in previous problems, and the modified wheel philosophy (2 wheels and teflon ball) was not the best scheme, we had many trouble to make the robot follow the line and move smoothly on the floor. The solution is simply to tune the parameters of the PID but since we had a small range of speed difference between the right and left wheel (i.e. this speed difference between the right and the left motors are the parameter making the robot turn) and since the robot cannot move as smooth as possible because of mechanical design, tuning parameters of PID was very painful.

S: We tried to achieve the best movement behaviour after many trials.

- **Problem and Solution to non-smooth Line follower**

P: Since we used magnets on the floor as a representation of the nodes (offices), and since the reed switches used to sense these magnets are mounted beneath the robot case, these magnets should be placed really close to the line. That brings up another problem which is the possible collision between the robot and the magnets while the robot is moving. That is an undesirable thing to us because it can make robot stop, go outside of the line etc.

S: To not allow this happening, we mounted reed switches on the back side of the robot and make the robot move as accurately as possible in the sense that two wheels at the back side would not collide with the magnets. That comes with tradeoff between the movement accuracy and smoothness. We chose the accuracy over the smoothness and tune the PID parameters according to that.

- **Problem and Solution to keypad not working**

P: The keypad that we bought from amazon did not work, even when it is connected to the pc.

S: Since we did not have enough time to order another one, we chose to discard this and try to simulate the interaction between the user and the robot (generally for password request purposes) by using a simple switch. Of course, this limited us from showing every feature of our system that we developed (e.g. we could not show the case of user entering a wrong password) but that is another tradeoff we had to decide; time vs. quality.

13.3. Future Work

In this semester, we have implemented basic server-side management, storage, communication, client-side user-interface and event-reaction, and robot-side locationing and picking up/delivering packages. We believe this to be an interesting topic and we would be definitely happy to extend our current design to a much more functional, error tolerant, and creative system. Below are some of our ideas that may be able to fulfill in the near future.

- **Multiple-platform support:** We plan to develop the Android application, and MacOS application of the client side. Currently we only have java application which can be well used in personal computer, as long as the computer has installed Java Runtime Environment. In the future we plan to extend the client terminal from computer to modern mobile terminals, such as Android phone, iPhone, and iPad.
- **Multiple-floor & Multiple-building:** We plan to make our system capable of delivering packages among different floors, and even different buildings. We have planned the ways of achieving this, such as making the robot take the elevator and choose which floor as it would like to go to, or using hinged crawler to go up and down the stairs.
- **Speech Recognition:** We think there might be multiple ways to book a delivery, or user authentication. One of them might be speech recognition. By installing video cameras, speech sensor, and motion sensor such as Kinect sensor, our robot can recognize what people are saying by there speech and gestures (see article [34] for reference). Hence, booking a delivery might just need to say a simple sentence to the robot "Give this package to John", then the delivering will start. The user authentication can also be done by the recognizing voice of the sender or receiver.

- **An Internet of Things:** The Delivery service might ultimately become an internet of things. Just as the article have stated, deliveries would be possible over a wide area using a series of hops and would be a physical implementation of the “packet switching” model that directs data across the internet [32]. The patterns of delivery might be diverse, no matter it’s using a robot that moves along the road, or a drone that flies in the sky. Yet these patterns have something in common, which are unmanned and multi-hopped. A small package can be delivered from China to United States by multiple unmanned delivery machines in several hours. Delivery machines will swap the package in certain nodes, give it to the other machines that takes charge of the next part of route, as well as refill power to start its next work. Then in several relays the package is delivered to the recipient without any worker in the middle of the process. We hope our auto delivery system can ultimately accomplish the system we depicted above.

BIBLIOGRAPHY

- [1] Android user interface. URL: <http://developer.android.com/guide/topics/ui/index.html>.
- [2] Arduino design. URL: <http://arduino.cc/forum/index.php/topic,6595.0.html>.
- [3] Arduino mega 2560 r3. URL: http://www.amazon.com/Arduino-MEGA-2560-R3/dp/B006H0DWZW/ref=sr_1_1?s=electronics&ie=UTF8&qid=1350484606&sr=1-1&keywords=arduino+mega+2560+r3.
- [4] Arduino motor shield. URL: http://store.arduino.cc/ww/index.php?main_page=product_info&cPath=11_5&products_id=204.
- [5] Arduino uno. URL: http://store.arduino.cc/ww/index.php?main_page=product_info&cPath=11_12&products_id=195.
- [6] Arduino wifi shield. URL: http://store.arduino.cc/ww/index.php?main_page=product_info&cPath=11_5&products_id=237.
- [7] Automatic identification and data capture. URL: http://en.wikipedia.org/wiki/Automated_identification_and_data_capture.
- [8] Automatic storage systems at nitco. URL: http://www.nitco-lift.com/custompage.asp?pg=automatic_storage_systems.
- [9] Code coverage. URL: http://en.wikipedia.org/wiki/Code_coverage.
- [10] Fata inc. URL: http://www.fatainc.com/gallery/main.php?g2_itemId=72.
- [11] Information security. URL: http://en.wikipedia.org/wiki/Information_security.
- [12] Internet of things. URL: http://en.wikipedia.org/wiki/Internet_of_Things.
- [13] Junit. URL: <http://en.wikipedia.org/wiki/JUnit>.
- [14] K&n warehousing and shipping. URL: http://www.knfilters.com/Warehousing_and_Shipping.htm.
- [15] Measuring junit code coverage. URL: http://www.cafeaulait.org/slides/albany/codecoverage/Measuring_JUnit_Code_Coverage.html.
- [16] Photointerrupter reflective phototransistor 4-pin tube. URL: <http://www.alliedelec.com/Search/ProductDetail.aspx?SKU=70026408>.
- [17] Robot learning technology for multi-task service robots. URL: <http://www.skilligent.com/products/documents/docs/Multi-Task-Service-Robots.pdf>.
- [18] Sainsmart l293d motor drive shield for arduino duemilanove mega uno r3 avr atmel. URL: http://www.amazon.com/SainSmart-L293D-Shield-Arduino-Duemilanove/dp/B00813HBBO/ref=pd_bxgy_e_img_y.
- [19] Smart robot design. URL: <http://wenku.baidu.com/view/30e72b6427d3240c8447ef91.html>.
- [20] Switch reed. URL: <http://www.alliedelec.com/Search/ProductDetail.aspx?SKU=70026408>.
- [21] Usb/ ps2 numeric keypad ps2/usb, htk-100. URL: http://www.amazon.com/Arduino-MEGA-2560-R3/dp/B006H0DWZW/ref=sr_1_1?s=electronics&ie=UTF8&qid=1350484606&sr=1-1&keywords=arduino+mega+2560+r3.
- [22] Wifi shield for arduino. URL: http://store.arduino.cc/ww/index.php?main_page=product_info&products_id=237.

- [23] An overview of 802.11 wireless network security standards and mechanisms, October 2004. URL: http://www.sans.org/reading_room/whitepapers/wireless/overview-80211-wireless-network-security-standards-mechanisms_1530.
- [24] Barcode, September 2012. URL: <http://en.wikipedia.org/wiki/Barcode>.
- [25] Barcode scanner, September 2012. URL: http://en.wikipedia.org/wiki/Barcode_scanner.
- [26] Database, September 2012. URL: <http://searchsqlserver.techtarget.com/definition/database>.
- [27] Denial of service attack, October 2012. URL: http://en.wikipedia.org/wiki/Denial-of-service_attack.
- [28] EclipseLink jpa implementation, November 2012. URL: <http://www.eclipse.org/eclipselink/>.
- [29] Event-driven architecture, December 2012. URL: http://en.wikipedia.org/wiki/Event-driven_architecture.
- [30] Facade pattern, October 2012. URL: http://en.wikipedia.org/wiki/Facade_pattern.
- [31] Gps, September 2012. URL: http://en.wikipedia.org/wiki/Global_Positioning_System.
- [32] An internet of airborne things, December 2012. URL: <http://www.economist.com/news/technology-quarterly/21567193-networking-enthusiasts-dream-building-drone-powered-in>
- [33] Introduction to java rmi, October 2012. URL: <http://www.javacoffeebreak.com/articles/javarmi/javarmi.html>.
- [34] Iuro robot finds its way through cities, with your help, December 2012. URL: <http://spectrum.ieee.org/automaton/robotics/humanoids/iuro-robot-finds-its-way-through-cities-with-your-help/>.
- [35] Java annotation, November 2012. URL: http://en.wikipedia.org/wiki/Java_annotation.
- [36] Java ee, November 2012. URL: http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition.
- [37] Java persistence api (jpa), November 2012. URL: http://en.wikipedia.org/wiki/Java_Persistence_API.
- [38] Jpa annotations, November 2012. URL: <http://www.oracle.com/technetwork/middleware/ias/toplink-jpa-annotations-096251.html>.
- [39] Master/slave pattern, October 2012. URL: [http://en.wikipedia.org/wiki/Master/slave_\(technology\)](http://en.wikipedia.org/wiki/Master/slave_(technology)).
- [40] Master/slave pattern (2nd reference), October 2012. URL: <http://www.ni.com/white-paper/3022/en>.
- [41] Master/worker pattern, October 2012. URL: <http://wiki.gigaspace.com/wiki/display/SBP/Master-Worker+Pattern>.
- [42] Modular programming, October 2012. URL: http://en.wikipedia.org/wiki/Modular_programming.
- [43] Monolithic system, October 2012. URL: http://en.wikipedia.org/wiki/Monolithic_system.
- [44] Multi threading tutorial, December 2012. URL: <http://www.comp.lancs.ac.uk/~weerasin/csc253/tutorials/week7.html>.
- [45] Multitier architecture, October 2012. URL: http://en.wikipedia.org/wiki/Multitier_architecture.
- [46] Non-functional requirements as user stories, September 2012. URL: <http://www.mountangoatsoftware.com/blog/non-functional-requirements-as-user-stories>.

- [47] Pulse width modulation, October 2012. URL: http://en.wikipedia.org/wiki/Pulse-width_modulation.
- [48] Shortest path problem, October 2012. URL: http://en.wikipedia.org/wiki/Shortest_path_problem.
- [49] Software coupling, October 2012. URL: [http://en.wikipedia.org/wiki/Coupling_\(computer_programming\)](http://en.wikipedia.org/wiki/Coupling_(computer_programming)).
- [50] Three tier architecture, October 2012. URL: http://en.wikipedia.org/wiki/Multitier_architecture#Three-tier_architecture.
- [51] User story, September 2012. URL: http://en.wikipedia.org/wiki/User_story.
- [52] Web server, September 2012. URL: http://en.wikipedia.org/wiki/Web_server.
- [53] Bennett Brumson. New applications for mobile robots, April 2012. URL: http://www.robotics.org/content-detail.cfm/Industrial-Robotics-Feature-Article/New-Applications-for-Mobile-Robots/content_id/3362.
- [54] Prof. Ezhan Karasan. Computer networking course cs421, lecture slides 4, October 2012. URL: <http://cs.bilkent.edu.tr/~terzi/cs421/>.
- [55] Smart Robot. The sr4 office robot, September 2012. URL: http://www.smartrobots.com/office_robot.php.
- [56] Aaron Saenz. Willow garage lets you control your robot from a web browser, April 2010. URL: <http://singularityhub.com/2010/04/12/willow-garage-lets-you-control-your-robot-from-a-web-browser-video/>.
- [57] Christina Hernandez Sherwood. Hospital tech: robots in the pharmacy and barcodes on the patients, December 2010. URL: <http://www.smartplanet.com/blog/pure-genius/hospital-tech-robots-in-the-pharmacy-and-barcodes-on-the-patients/5072>.
- [58] Freight transport. Put that in your pipe and poke it, January 2011. URL: <http://www.economist.com/node/17848523>.
- [59] Gali Weinreb. Israel invests 11m dollars in nano drug delivery robot, July 2012. URL: <http://nocamels.com/2012/07/israel-invests-11m-in-nano-drug-delivery-robot/>.
- [60] Wikipedia. Inductive charging. URL: http://en.wikipedia.org/wiki/Inductive_charging.

A. EXISTING COMMERCIAL SOLUTIONS

A.1. Traditional system

Traditionally, as stated in the problem description, deliveries have been done in a non-automated manner: sometimes there is a person in charge of this problem and sometimes the sender is just simply forced to go and deliver the document to the recipient. Of course, this method is not scalable and can become very inefficient.

A.2. Integrated delivery system using pipes

Accordingly, several approaches have been made to solve this inefficiency. One of this approaches is to integrate the delivery system in the building using, for example, wide pipes [58]. Some examples of this can be seen in a great variety of places (such as hospitals, city councils or police buildings). This system, of course, have several limitations. On the one hand, it needs to be designed along with the building, which is not cheap nor convenient, because many departments may need to deploy the automation in an already existing building. On the other hand, this kind of delivey service does not allow packages nor inter-building communication. Finally, the user interface in these systems is a box in each room with a certain amount of buttons.

Our project overwhelm this costs and, even more, will allow the costumer to scale the solution to several buildings (this feature will be done in a future deployment of the project). Our project will provide simpler and friendly user interfaces than annoying boxes in every room, because today all the rooms have computers, and therefore they can be used as an *existing* way of dealing with the system. On the other hand, our user interface will be more interactive than the existing one, because the user will be able to ask for feedback about how the delivery is going on.

A.3. Drug delivery using robots

As we previously state, our working problem applies specially on the hospital environment.

Autonomous mobile robots (AMR) can play a role in assisting doctors in medical procedures, such as drug delivery, or the development of mobile treatment systems for specialized equipment. However, the cost of an AMR is huge and the moving algorithm inside the robot is very complicated. A nanometer-sized drug delivery systems cost more than \$11 million [59]. Compared with AMR, our automatic delivery car cost much less, uses simpler algorithm, and has a feature of easily been integrated.

As the system is a one-way delivery (from the nurse's station to ward, since the patients will do not order drug delivery to others) and each ward is a stop, the user interface is nothing else but two big buttons on top of the robot. After the nurse put drugs in robot, he or she presses the green 'Go' button, and it proceeds on its route. If the nurse needs more time, he or she can press the red 'Stop' button. Each ward acts as a stop, and the robot starts from the nurse's station and stop at each ward. Therefore, the delivery is not done in the front of each pacient room

From the interface point-of-view, our system have two clear advantages in comparison to the drug delivery existing system:

- Our system can deal with the delivery request among the employees
- The UI can be more vivid, such as a website or an Android App. This allows powerful tools to select usual/favorite destinations and this helps improving the ease-of-use of the application.

A.4. Automated warehouses

Although our working problem might seem similar to automated warehouses. The solutions applied there (we can see manufacturers in [8] or [10] and an implementation on [14]), however, are not well sized for small packages, and in case it were, automatic warehouses uses complex translation mechanisms to accelerate delivery, which is not needed in our case. Thus, the customer will see that the existing solutions and techniques for warehouses can't be applied to our problem.

B. INTERACTION DIAGRAM FOR UC-8 DELIVERY

To begin with, we are using three-tier client-server software architecture model to logically separate the presentation, application processing and data management (reference [45]). The signs of this design choice can be seen in every interaction diagram of the use cases given in the followings. Additionally, another design choice that we made is to separate the overall system into two stacks; server and robot.

The related objects used in use case Delivery can be found in (table B.1). As can be seen there is not any objects related with presentation tier for this specific use case. All necessary action will be done for this use case by the objects in server and data layers.

This use case is the black box containing the most important functionality of the system so we gave a lot of care and attention while designing. First thing we considered, our design should allow the least number of interactions because the optimization of the number of interactions for this use case is crucial in the sense that it can provide a clear and more scalable implementation. Secondly, we tried to make it to answer the needs of our application. To explain more, as we specified in the requirements part (Section 2.2) booking a delivery should be done in a seamless and robust manner. Therefore, we tried to make possible failure cases anticipated by the system. Also, the work sharing between the objects is carefully one but of course as the case for all design problems some important parts become responsible for more work. Since the logic itself and logical interactions between the objects included in this use case are complex, we tried to specify one "facade" (reference [30]) assigned for the objects of logic tier in two parts. As can be understood from its name one of them is ServerCommunicator and the other is RobotCommunicator which manage the communication between logic objects and data objects in their respective stacks and also they are responsible for the interaction between the two stacks.

BookDelivery Interaction General Diagram The interaction diagram for this use case includes 2 main tasks which are as follows; 1. Get the most prior pending delivery 2. Schedule the robot action for this most prior pending delivery. This action includes notifying the receiver about the delivery and moving the robot its next destination. Also, once the robot reaches the next destination and if this is the target address of the delivery, take care of delivering the correct packets to the receiver in the case of delivery type is 'hand in' or picking up the packets from the correct receiver in the case of the delivery type is 'pick up'.

Note: We separated the delivery into two types; one is 'hand in' and the other is 'pick up'. We used the delivery as the encapsulating phrase for both types because it sounds the most logical choice for our project design. In that sense, delivery is describing the task of both picking up the packages from the sender and handing in the packages to the receiver.

As one can see from the interaction diagram of this part (figure B.1), there are some function calls between the objects which are like dark black boxes. These functions (can be seen in the interaction diagram encircled by blue rectangles) are shown without the inner workings in this figure to keep things less complicated and more readable. In the following parts the inner workings design of them will be revealed.

Get the most prior pending delivery: This part is crucial for the success of the following operations. The inner workings can be explained as follows (figure B.2); Once the automatic delivery system is started ServerController will automatically create the DeliveryController which is responsible for checking the pending delivery list, getting the most prior delivery. Checking delivery list is done by simply listening the size of the pending delivery list. Once the size of the delivery list is not '0', DeliveryController fetches the pending delivery list and initiates the `getMostPriorDel()` function which basically runs an algorithm to get the most prior delivery to be taken care of (Main_interaction_diagram). This most prior delivery decision is done according to the delivery urgency status entered by the sender and the location of the

delivery 'pick up' and 'hand in' address locations. For each delivery in the delivery list, distance of the delivery target address from the current position of the robot is received from FloorMap and then by simply choosing two weighting variables for the distance and urgency of the delivery ('a' and 'b' in the interaction diagram) a constant for each delivery is calculated by $\text{constant} = \text{distance} * a + \text{urgency} * b$ (MostPriorDelInteractionDiagram). These variables can be chosen to make urgency or distance as the significant factor while deciding the most prior delivery. This type of approach allows the system to modify this decision process easily. Then by choosing the delivery corresponding to the least constant, the system gets the most prior delivery in the list.

Schedule robot action according to the pending task:

After the most prior pending delivery is got then the ServerController needs to schedule the necessary robot action. This part of the interaction diagram is complicated and large, to make it visible we tried to separate this part into two and connect these two parts by a blue line (figure ??). This part includes the followings; (1, 2 and 3 can be found in (figure B.1) and the rest in (figure B.3))

1. If the delivery type is 'hand in', then ServerController will notify the receiver of the delivery. This notification will include the sender's information and also the estimated arrival time of the robot to the receiver's office. (The calculation of the estimated arrival time will be done by using simply the distance between the current robot position and receiver's position.)
2. No matter which type of delivery is fetched as the most prior delivery, the robot will be moved to the next position (position refers to 'node' which are simply the magnets deployed on the floor). The determination of the next position will be done by ServerController by using FloorMap. We left this part as a black box right now. Once we move to the implementation of the robot stack, we will come back to this.
3. Once the robot moves to the next position, the algorithm to find the most prior delivery will be run to see whether there is a related pick up or hand in associated as the target address to this position. This is done by first finding again the most prior delivery in the pending delivery list (same procedure with the previously explained one). Then if the current position of the robot and the target address of the most prior delivery are same, corresponding action (handlePriorDelivery(pendingDelivery) (figure B.1) is handled by the ServerController.
4. The following actions will be taken to accomplish 'handlePriorDelivery(pendingDelivery)'
 - a) The robot's buzzer is ringed until the user makes an interaction with the robot. If the specified timeout is reached while the robot is waiting for the user, a notification will be sent to the ServerController and the operation will be terminated. In that case, system will begin operation from the top of the interaction diagram. (see loop encapsulating all interactions (figure B.1))
 - b) The other case is that user enters his/her password before timeout occurs. This entered password is then sent to the ServerController for checking. Password checking is actually done by PasswordChecker which is called by the ServerController. And again by the same communication line, the user is informed about the result of the password check (for clarity and simplicity, we omitted the unsuccessful password entry attempts and password reentry mechanism for this report. We tried to add the important interactions of the important functionalities).
 - c) Once the password is confirmed,
 - If the delivery type is 'hand in', the related tray will be opened by the robot.
 - Else if the delivery type is 'pick up', a free tray will be opened by the robot.

For both of the cases above, ServerController will check the SecureBoxContents to open the related tray and wants the SecureBoxContents object to update itself according to the tray opening action. Once the correct tray is opened, robot waits for the user to finish its package submission or package retrieval. After that the robot sent a 'Delivery is successfully completed' notification to the ServerController. Then ServerController asks the PendingDeliveries to remove the completed delivery from its list and system goes back to top of the interaction diagram.

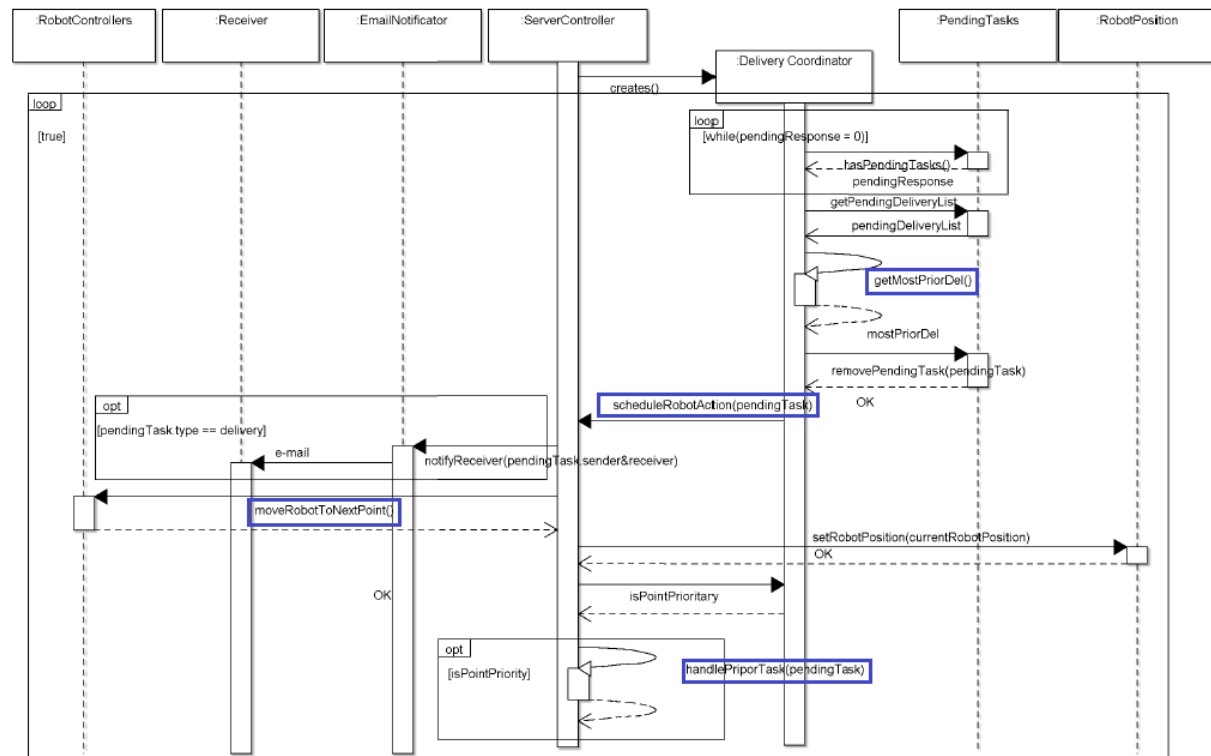


Figure B.1.: General View of the Delivery Use Case Interaction Diagram.

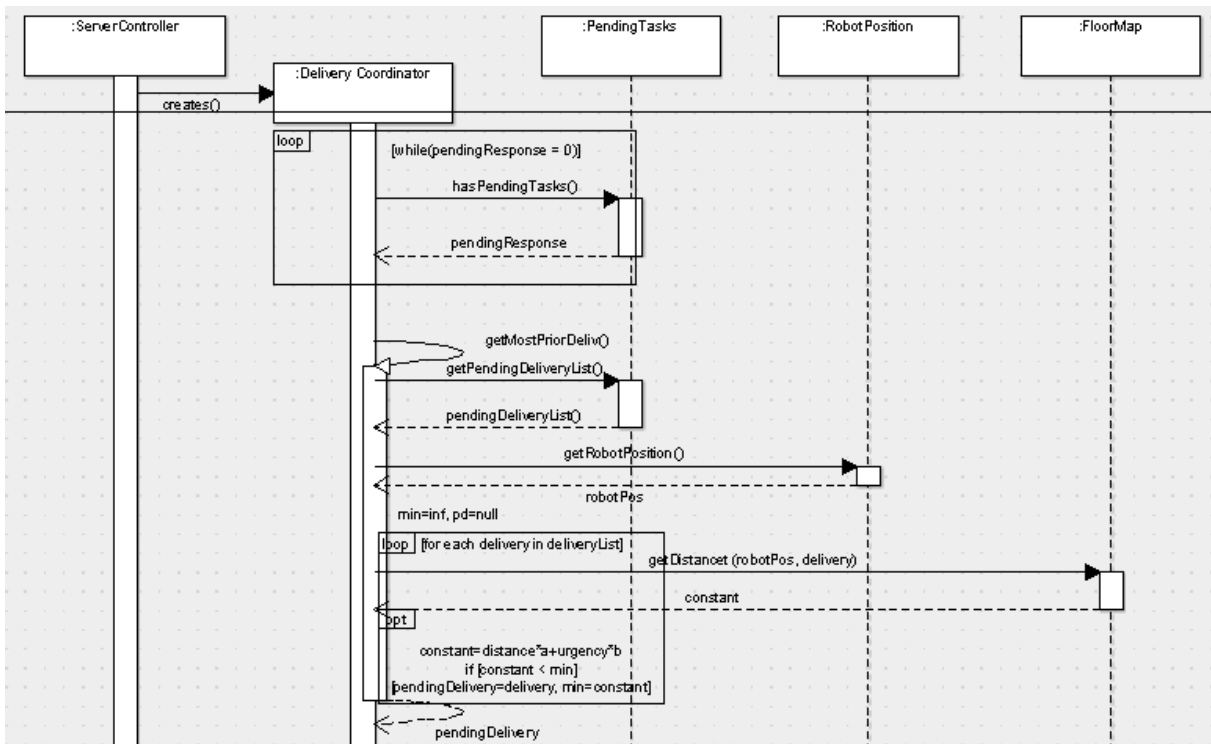


Figure B.2.: Expanded view of the upper part of the main interaction diagram for Delivery use case which shows getting the most pending delivery by the server controller.

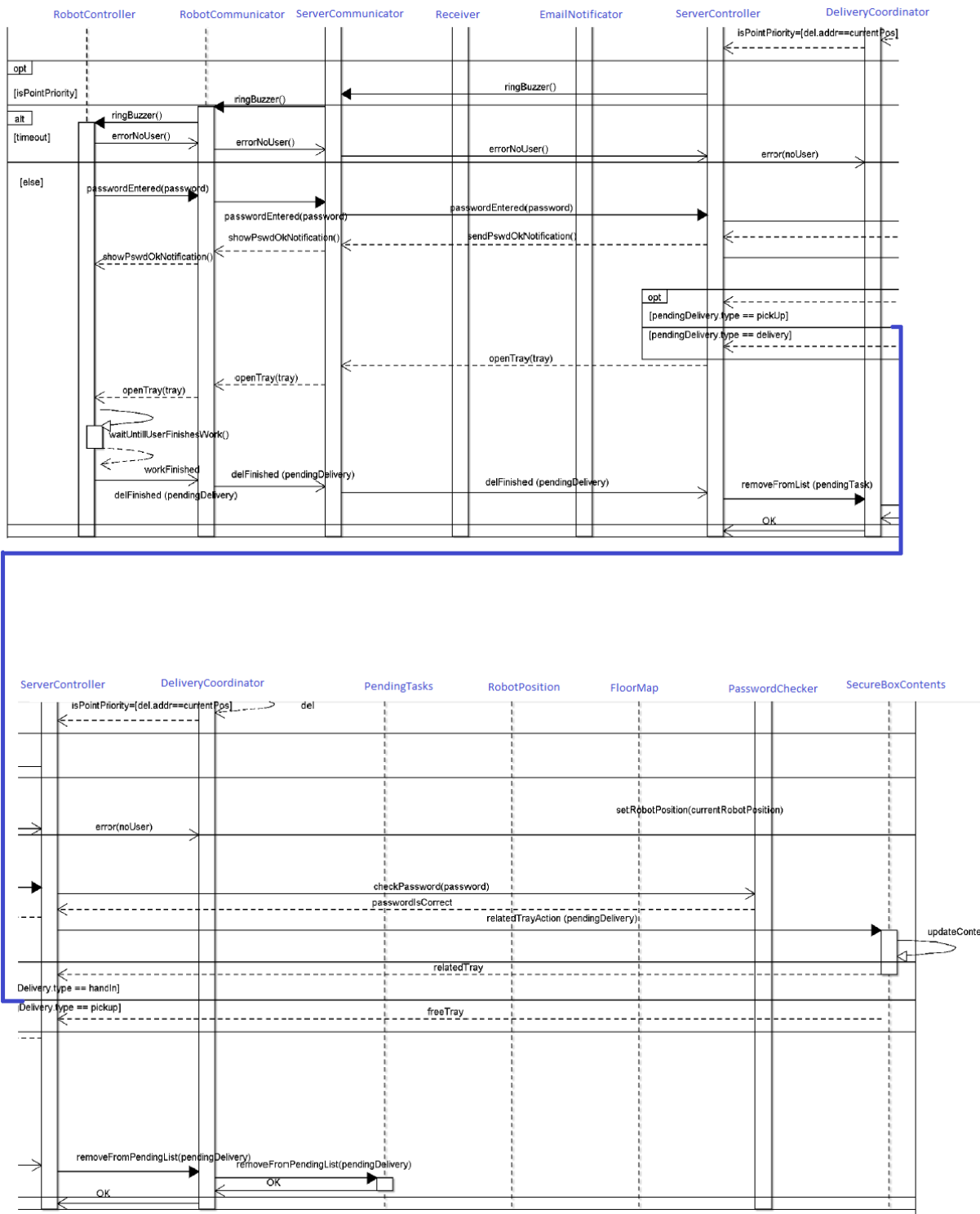


Figure B.3.: Expanded view of the down part of the main interaction diagram for Deliver use case which shows the inner workings and interactions after the ServerController gets the most prior pending delivery. It is mainly stating the inner workings of the black box handlePriorDelivery(pendingDelivery) in (figure B.1)

Design Layer	Server Part Objects	Robot Part Objects
Presentation Tier	-	-
Logic Tier	Server Controller	RobotController
	DeliveryCoordinator	RobotCommunicator
	ServerCommunicator	
	PasswordChecker	
Data Tier	EmailNotificator	
	FloorMap	RobotPosition
	SecureBoxContents	
	PendingTask	

Table B.1.: Showing the objects created for the Delivery use case and their associated architecture tiers.