|  |
| --- |
| FONTYS UNIVERSITY OF APPLIED SCIENCE |
| Technical Manual |
| Parcel Handling Simulation |
|  |
| **GDS - Group 3** |
| **11/24/2010** |

|  |  |  |
| --- | --- | --- |
| Document name |  | Technical Manual |
| Project name |  | Parcel Handling Simulation |
| Department |  | ICT |
| Client |  | Peter Boots |
| School tutor |  | Casper Schellekens |

|  |  |  |
| --- | --- | --- |
| Group leader |  | Qian Li |
| Group members |  | Antoine Girard |
|  |  | John Ibeagha |
|  |  | Kritian Kolev |
|  |  | Sebastien Lepage |

**Table of Contents**

[1. Introduction 1](#_Toc283026673)

[1.1 Background 1](#_Toc283026674)

[1.1.1 Project Overview 1](#_Toc283026675)

[1.1.2 Document Overview 1](#_Toc283026676)

[1.2 Objective 1](#_Toc283026677)

[1.3 Target Group 1](#_Toc283026678)

[2. Design 2](#_Toc283026679)

[2.1 Class Diagrams & Functions 2](#_Toc283026680)

[2.1.1 Airport Library Class Diagram 2](#_Toc283026681)

[2.1.2 Handling Simulation Classes 3](#_Toc283026682)

[2.2 Sequence Diagrams 4](#_Toc283026683)

[2.3 Algorithms 4](#_Toc283026684)

[3. Open Issues 9](#_Toc283026685)

# Introduction

## Background

### Project Overview

The parcel handling simulation is a distributed application that is intended to resemble the parcel handling simulation at the airports. It includes the simulation of the conveyors used to transport the parcels and baggage, the inputs, which are the check-in and parcel drop-off desks at the airport, the sorters that route the items to their destinations and the outputs, which are the destination gates of the parcels.

The application is fully implemented with all required functions. The users will be able to build the conveyors as they want by drawing lines on the working area. Each parcel will have a set of information including as a destination (plane/destination gate), ID, a priority based on urgency of delivery, etc. which would allow the sorters to navigate the parcel to its destination. The information of parcels and planes are generated by the users before running the simulation. Moreover, properties of storage capacity and belt speed can also be generated by the users.

### Document Overview

The present document contains the design of the Parcel Handling Simulation application and its main technical parts. The class diagrams and sequence diagrams are introduced in the next chapter, along with all the classes and their related functions. Also the algorithms of the sorters will be described.

In the following chapter, some of the codes will be explained which contains the instructions to make changes for adding extra functions or modifying existing functions.

In the end, open issues will be discussed for further development.

## Objective

This document is intended to state the technical instructions for the project development and maintaining, which will help to improve the quality of the application and provide guidance when a problem occurs.

## Target Group

The current manual is set up for the software engineers who are responsible for monitoring and maintaining the application when it is put into use in real life.

The following IT knowledge is required to fully understand the current document:

* UML – Unified Modeling Language
* C# Programming
* .NET Remoting

# Design

The Parcel Handling Simulation system is a distributed system and includes 2 main parts: Client side, Server.

**Client side:** This is the system where the users can connect to the server and handle the simulation.

**Server:** This handles the main simulation configuration, on that configuration based; the client will run the simulation according to the configuration from the server. The server have an interface where conveyor speed, parcel interval generation, storage maximum luggage and regarding the server configuration such as port, end point address, can be shared with other clients.

## Class Diagrams & Functions

This chapter describes all the classes and methods which are included in our class diagrams. The Airport Library class diagram shows the classes of the components and methods that the client needs to create a simulation while the Handling Simulation class diagram is intended to explain the methods to handle the simulations such as making a new simulation, saving a simulation and etc.

### Airport Library Class Diagram

**AirportZone** - This class is one the server side and is to take charge of creating the simulation interface of our application. There are 4 classes inherited from it which are DestinationGate, Storage, CheckIn and Sorter. In this class, the method OnRender with a return type of void is to paint all the components on the interface.

**AirportArea** - This class is on the server side and is to take charge of the activities on the airport zone, which is also medium of painting conveyors and transferring the image on the airport zone. All methods in this class are of type void except method FindAirportZone. CreateTable and ResizeTalbe are functions used to create a new simulation interface and resize the interface area which have no parameters. PopulateTable is to put the new table in the table list in the system, which has a parameter namely List<AirportZone> zones while SetEventsToZone is a function to set all the events into the new interface area like DragDrop, DragEnter, MouseDown, MouseMove with a parameter called AirportZone zone. FindAirportZone has a function of locating where the mouse is pointing at in the interface area and it has a return type of void.

**DestinationGate** - Inherited from AirportZone, this is a class used to add a destination gate on the airport area and connected to the class Flight and Storage. Each destination gate has storage. And eachclass is in charge of the flight information including which gate the plane will be waiting and the departure time of every flight.

**Storage** - This class is in charge of the storage information which is also related to Class Luggage. It contains the luggage waiting for its flight and the capacity.

**Luggage** - This class takes charge of all luggage with the plane/destination gate it goes to.

**CheckIn** - This class is in charge of the check in gates. The luggage enters the simulation from the check in gates and the luggage information will be generated.

**Conveyor** – This class manage access between two component herited from AirportZone such as CheckIn, Sorter and DestinationGate.

**Way** – This class manage a set of conveyor aggregating a path between a CheckIn and a DestinationGate.

**Sorter** - This class is to take charge of all the sorters in the simulation. It contains the algorithms and methods to identify the luggage ID and configure the route for the luggage to go to its right destination gate/storage.

*See more details on the* [*technical msdn documentation*](GDSg3%20-%20LibraryDocumentation.chm) and have a look on the class diagram in Visual Studio.

### Handling Simulation Classes

**HandlingSimulation** - This class is main class in this diagram, which is in charge of the user interface for the client to manage the simulation such as Open a simulation, Save a simulation and etc. It is on the client side, containing all the methods that the user could do with the simulations. There are 2 methods with a int return type while the others are with a void return type, and all the methods in this class has no parameter. Method AirportZoneIsSelected is a function to check whether a simulation interface is selected or not. DropNewAirportZone is a function used to create a new (empty) simulation interface for the system after the user calls the method NewDocument, which is creating a new simulation. Likewise, SimulationClosing is for the system to close the simulation interface after the user calls the method Exit. OpenFile is a function for the user to open an existing simulation while SaveDocument and SaveAsDocument is to save a simulation file. FileCreated, FileChanged and FileRenamed are methods that executed after the SaveDocument and SaveAsDocument methods are called. The only 2 methods with an int type are Serialize and Deserialze. These 2 methods are used to serialize the position of components on the airport zone by the point on the interface.

**Navigation** - This class is used for the user to navigate the simulation. It is on the client side. Navigation contains 3 methods: Pause, Play, Stop so the user can start, pause or stop the simulation. The methods here are of type void and have no parameter. The methods are created as buttons of Actions on the interface.

**GridProperties** - This class is in charge of editing the properties of components in the simulation. The methods are used to display the properties.

**ItemBox** - This class takes charge of adding components to create a simulation. It contains methods to drag the components into the zone and move the components inside the zone.

**OutputControl** - This class is in charge of checking whether a simulation is complete. It includes methods which check if there are conveyors between 2 different objects and whether the simulation is able to run. The errors will be shown in the list to remind users to modify the simulation.

**Configuration** – This window permit to configure the packages and the network system.

## Sequence Diagrams

Most important and interesting part of the work is the simulation and the algorithm used. First, let’s introduce the problem. To recall, the server side take care about the configuration of the simulation running on the client side. This configuration understands:

* Conveyor speed
* Maximum Parcel interval (Manage by Sorter)

If this configuration is lower, it means if maximum parcel interval from the configuration on the server side is lower than the maximum parcel interval from the configuration on the client side, so there is overloading. In this case, from existing packages, new packages are created and are added at the end of the line luggage.

## Algorithms

In a general way, let’s see the route calculation function which permits to update existing packages and run the graphic simulation for more details, to generate on Visual Studio or another reverse engineer software:

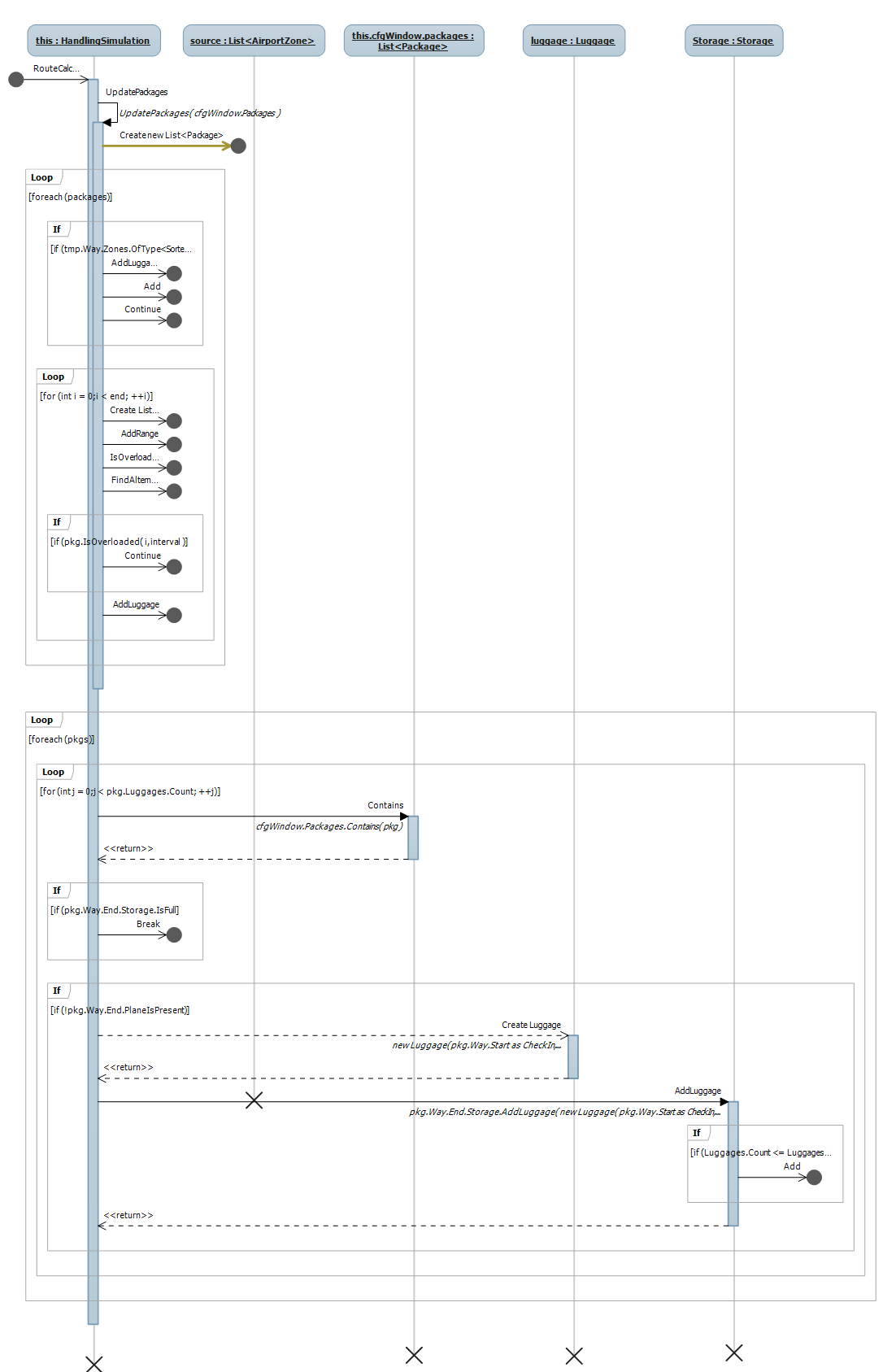


Figure 1 Route Calculation Algorithm

Let’s check now, the main functions inside the RouteCalculation function:

* UpdatePackages: Create new Package if there is an any overloading, and add them at the end of the simulation
* FindAlternative: If a Package is overloaded, try to find an existing and free Package.

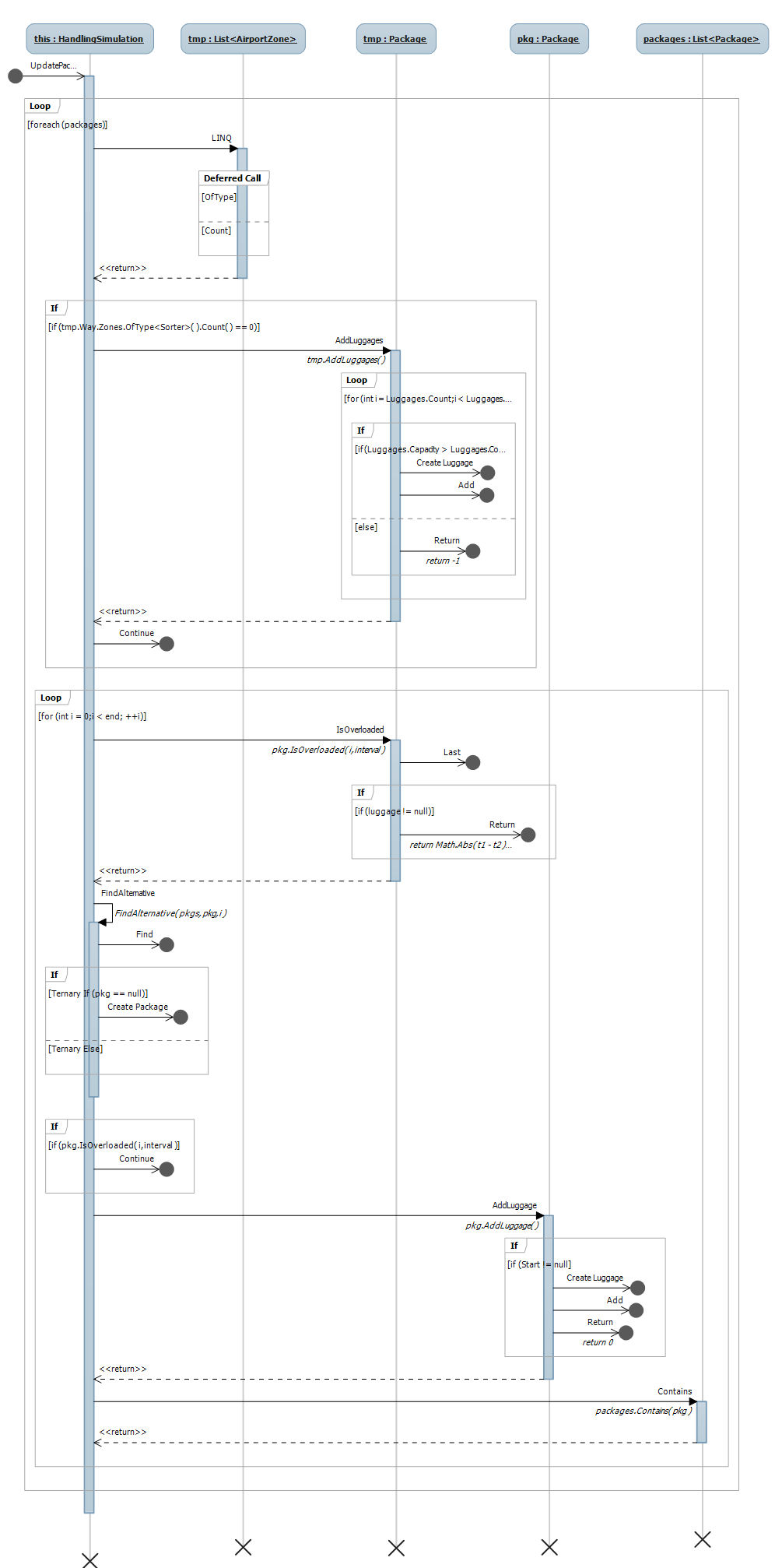


Figure 2 Update Packages Algorithm

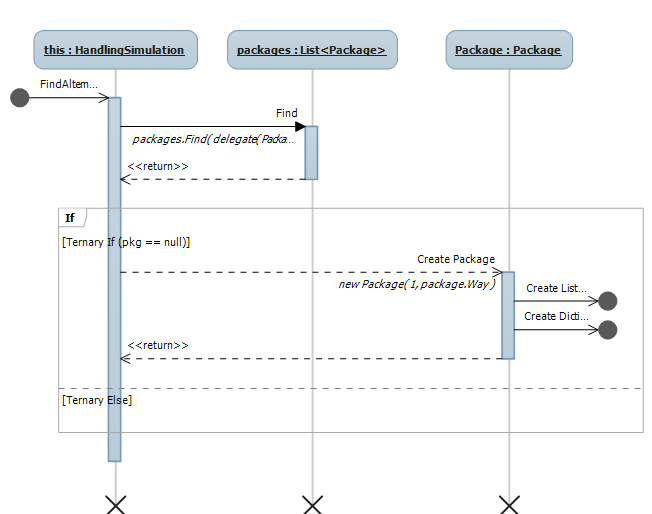


Figure 3 Find Alternative Package Algorithm

Once all of these algorithms are executed, it’s then possible to run the simulation (if the server is running).

# Open Issues

According to the specifications, there are no open issues; all the functionalities have been successfully implemented. Please see the User Specifications Requirements.