ECE 568 ERSS: Final Project

Mini-Amazon / Mini-UPS

Protocol Document

Group 4

An Qin, Ruiqi Li, Ang Xie, Xunyu Chu, Feng Wang, Yijia Li, Yixin Zhuang, Fiona Ye, Nathan Tian

Language and Framework

In this project we will use Python, C++ and Java as programming languages. And we will use django as the web framework.

Interface

The communication of Amazon, UPS and the world simulator Must be based on Google Protocol. Each server MUST have a Google protocol buffer to receive messages.

Terminology

- World: World is where orders take place. There can be many worlds, each world is specified by its unique world id. It can be created by the UPS by sending a UConnect request leaving the world id blank. In a world, there are addresses which are represented by coordinates of integers.e.g.(2,4). There are warehouses controlled by Amazon and trucks controlled by UPS.
- To connect to a world, first send A/UConnect request with world id. After receiving a “connected” message, it means connection is on and you can start to send other requests. Notice: only one Amazon and one UPS is allowed to connect to a world at the same time.
- Simulation speed: Simulation speed is an option of the A/U command. It represents how quickly things happen in the world. The default value is 100 and it will be consistent before you change it. Once you change it, it will only affect future events. Notice: This is only used for testing/ debugging, programs should work at any speed.
- Disconnect: Disconnect is another common option of A/U command. Once you set it true, the server will finish processing its current job then, send a response finished = true and close the connection.
- Ack number: A/UCommands and A/UResponses have ack numbers to avoid losing messages. Each command contains a sequence number which is incremented on Amazon and UPS side. Once the world simulator finishes processing the request, it will send back a response with acks of those sequence numbers. Once you receive a response from the simulator, you should also return an ack back.

Interaction Scenarios

a. Amazon->UPS: Amazon sends `request_truck()` to UPS to request a truck to a warehouse with a given whid. The request contains shipId, whid and destination.
b. UPS->World: send `UGopickup()` to the world to request a truck.
c. UPS->Amazon: After receiving `UFinished()` from the world, UPS sends `truck_arrived()` to Amazon, which contains shipId and truckId.
d. Amazon->world: After receiving `truck_arrived()`, Amazon sends `APutOnTruck()` to the world and waits for `ALoaded()` from the world.
e. Amazon->UPS: send `ready_for_delivery()` to UPS, which includes shipId.
f. UPS->world: send *UGoDeliver*() to world.
g. UPS->Amazon: send `deliver_started()` to Amazon, which includes shipId.
h. Amazon->UPS: may send `change_destination()` to UPS to change the destination.
i. UPS->Amazon: `deliver_started()` will return as the response to changing destination.
j. UPS->world: received *UDeliveryMade()* from world.
k. UPS->Amazon: send `delivered()`, which includes shipId.

Protocol Definitions

Amazon SHOULD use request_truck() request to initiate a new delivery with UPS.
A request_truck() request MUST contain shipid, whnum, destination location coordination, and MAY have ups_account.
UPS SHOULD send *UGopickup()* following the receipt of request_truck().
When receiving *UFinished()*, UPS MUST send *truck_arrived()* to Amazon. Received Apacked(), Amazon SHOULD send *APutOnTruck()* to the world.
Before loading the shipment, Amazon MUST have packed the products to be loaded and delivered.
After receiving *ALoaded()*, Amazon SHOULD send *ready_to_deliver*() to UPS.
UPS starts the delivery by sending *UGoDeliver*() to the world.
Meanwhile, It SHOULD send *deliver_start*() to Amazon.
After receiving *UDeliveryMade()* from the world, UPS MUST send *delivered_success()* to confirm the completion of the order.

Message Format Specification

```
A -> U
message request_truck{
      required int shipid = 1;
      required int whnum = 2;
      optional string ups_account_id = 3;
      required int location_x = 4; //destination
      required int location_y = 5;
}
message ready_for_delivery{
      required int shipid = 1;
}
message change_destination{
      required int shipid = 1;
      required int location_x = 2; //destination
      required int location_y = 3;
}

// [START declaration]
syntax = "proto2";

message world_info{
    //After receiving world_info from UPS,
    //amazon must try to connect to the world
    //if connected successfully, response with
    //the world id and result true
    //otherwise, response with the world id and
    //false.
      required int64 worldid = 1;
    required bool result = 2;
}
//all the following interaction must wait until
//the world id is properly set up.
```

```
message request_truck{
      required int64 shipid = 1;
      required int32 whnum = 2;

      optional string ups_account_id = 3;

    required int32 location_x = 4; //destination
    required int32 location_y = 5;

    required string item_desc = 6;
}

message ready_for_delivery{
      required int64 shipid = 1;
}

message change_destination{
      required int64 shipid = 1;
    required int32 location_x = 2; //destination
    required int32 location_y = 3;
}

message AToU{
      optional world_info worldInfo = 1;
      optional request_truck request = 2;
      optional ready_for_delivery readyForDelivery = 3;
      optional change_destination changeDest = 4;
}

// [END messages]
```

U -> A

```
message truck_arrived{
      required int shipid = 1;
      required int truckid = 2;
}
message deliver_started{
      required bool status // true = success; false = fail
      required int shipid = 2;
}
message delivered{
      required int shipid = 1;
}
```

```
// [START declaration]
syntax = "proto2";

// [START messages]
message world_info{
      //This is used on initiation
      //a world is created on UPS side and
      //send to amazon.
      required int64 worldid = 1;
      //a response with worldif and result field will
      //be returned.
      //if the result is false, create a new world and
      //repeat above, until true is received.
}

//all the following interaction must wait until
//the world id is properly set up.
message truck_arrived{
      required int64 shipid = 1;
```

```
      required int32 truckid = 2;
}

message deliver_started{
      required bool status = 1;
      required int64 shipid = 2;
}

message delivered{
      required int64 shipid = 1;
}

message UToA{
      optional world_info worldInfo = 1;
      optional truck_arrived truckArrived = 2;
      optional deliver_started deliverStarted = 3;
      optional delivered delivered = 4;
}
// [END messages]
```