Drone Package Delivery System Design Document

Date: December 19, 2014
Author: Roland L. Galibert
Reviewer: Orr Inbar

Contents

Introduction

This document defines a design for implementing the services (DroneDeliveryMediator, DeliveryManagement and Drone Management) required to operate the overall Drone Package Delivery System described in the requirements document. In order to better identify the functionality these services must provide and how they should be structured, the document also includes expected use cases for the system, system process flows (sequence and activity diagrams) as well as a rough design of the basic user interfaces required to maintain the system.

Overview

The use of automated drones to deliver packages is an innovative technology whose practical application is already a reality. The German logistics firm DHL recently used a so-called “parcelcopter” to make test deliveries to islands off German’s northern coast, and Amazon and Google also have plans to use the technology in test situations.

The advantages of the technology is that it can be used to deliver a variety of package types quickly, automatically and economically. Drone can negotiate routes that delivery trucks or large airplanes cannot. As the technology is expected to improve even more, delivery through drone aircraft is likely to become an important option in logistics.

A system that can take advantage of the technology to provide a reliable way to deliver packages and that also has the ability to respond to the almost-certain rapid changes in the technology could certainly be of great financial benefit its operators. The Drone Package Delivery System is intended to be such a system.

Requirements

The requirements document specifies five sub-systems for implementing the Drone Package Delivery System. This design document focuses on the DroneDeliveryMediator, DeliveryManager and DroneManager sub-systems described in the document. The UserInterface sub-system is described in general detail, with the description provided mainly for the purpose of identifying the methods that should appear in the first three sub-systems. Finally, the fifth sub-system, the AuthenticationService, has been implemented in a previous design stage and this document will not cover the specifics of its design.

<u>DeliveryManager</u>

This sub-system is the front end of the overall system. It allows customers and admins to create and schedule deliveries. The following information is required for a delivery:

- Scheduling information – pickup time, expected delivery time, delivery priority
- Package information – dimensions (height, width, length), weight, delivery notes
- Routing information – pickup location (latitude, longitude), destination (latitude, longitude) receipient name, sender name

This sub-system must also capture post-delivery information, such as actual delivery time and if the package was lost so that an appropriate result can be tracked.

<u>DroneManager</u>

This sub-system maintains most of the inventory for the overall system (in the form of actual drones and hangers to house and maintain those drones) and also provides functionality for basic drone communication. Although the DroneDeliveryMediator sub-system below issues commands for drone destinations (i.e. to delivery or to hangers), the DroneManager handles the actual communication and also deals with changes in drone operational status.

The DroneManager must be able to support the following information and functionality:

- Drone information – name, type, fuel level, system status (guidance system and commlink), current location (latitude and longitude), destination (latitude and longitude), IP address of associated ground communication link.
- Drone type (a general type to be associated with each drone, such as Normal, SuperSonic, HeavyCarrier) information – name, fuel burn rate, maximum allowable package dimensions (height, width, length), maximum payload, speed classification
- Hanger – name, maximum capacity (drones and fuel), current capacities, operating budget (maximum and allocated)
- CRUD functionality for the above types.
- Drone communication functionality.

<u>DroneDeliveryMediator</u>
As its name implies, DroneDeliveryMediator exists between the two sub-systems above and is responsible for scheduling deliveries by assigning these to drones. The DroneDeliveryMediator centers on its automated control system (ACS), which automatically schedules and initiates deliveries.

The DroneDeliveryMediator must provide the following functionality:

- Scheduling of deliveries, ideally having the option to select between routing/scheduling strategies.
- Assigning deliveries to alternate delivery methods (e.g. ground shipping) if they cannot be delivered by drone
- Maintenance of available ground communication links.
- Overall monitoring of operating resourcese.
- Maintenance/persistance of system messages and alerts.

Use Cases

The Drone Package Delivery System is automated for the most part, with user interactions being mostly limited to the creation of deliveries, maintenance of inventory and system monitoring. Deliveries may be created either by external users or by Drone Package Delivery System admins on behalf of external users, while maintenance of inventory and system monitoring will naturally be carried by admins only.

In addition, it should be noted that most or all inventory maintenance will be limited to DroneManager service functions. The requirements require that the DroneDeliveryMediator be monitored, but as the system is mostly automated, monitoring should be limited to responding to alerts, such as missing drones.

(Package senders and recipients will be responsible for physical, non-system related tasks of setting up nets at sender/destination sites and (in the case of senders) for ensuring that packages are properly positioned at the expected pickup time).

Based on what I found on the UPS website, external users may register with UPS as users, but are not required to, so the Drone Package Delivery System will make that assumption as well.

The use cases will then follow the following sequence:

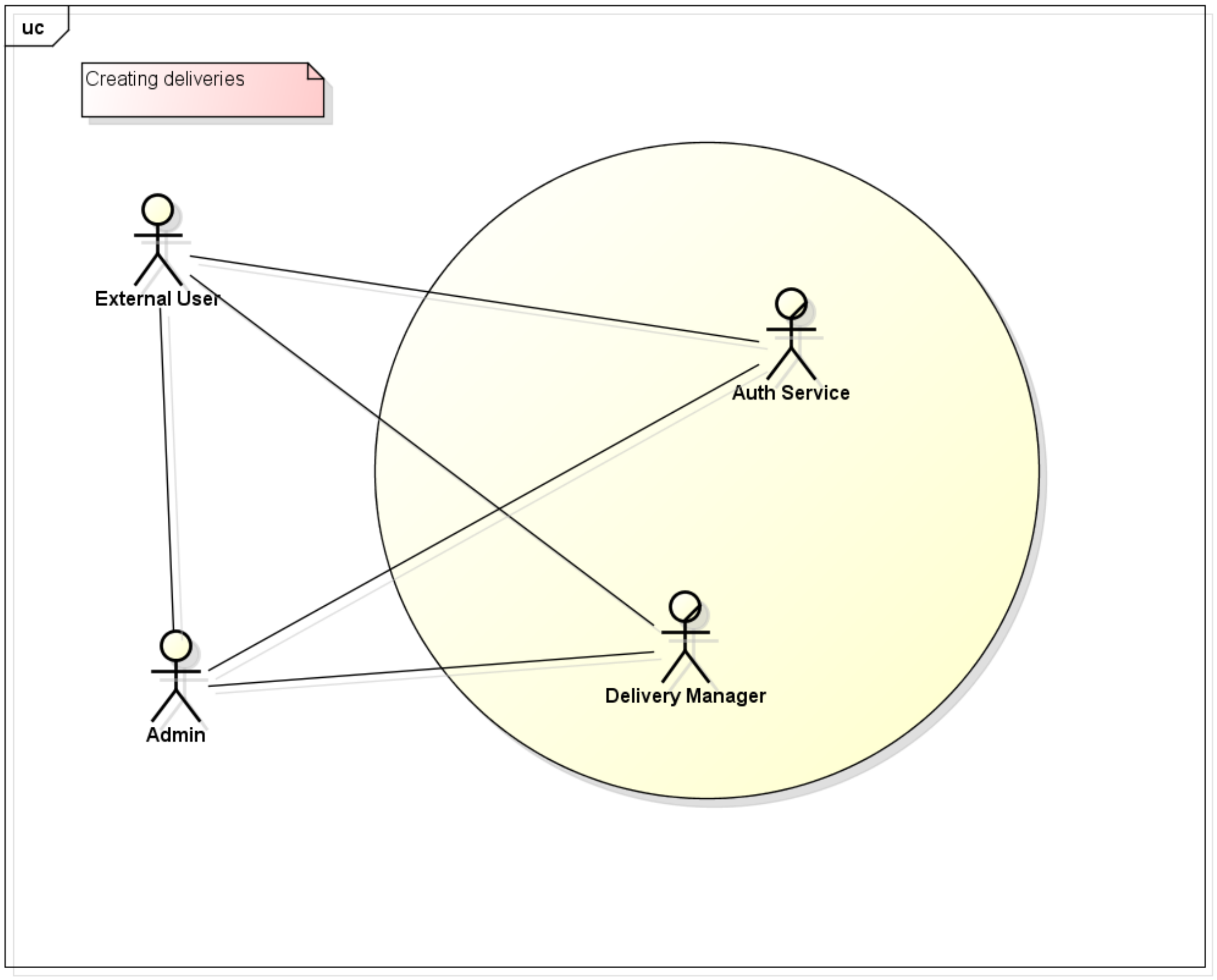
Delivery Creation (user)
1. User creates a delivery

Delivery Creation (user)
1. User registers with Drone Package Delivery System.
2. User logs in and creates a package
3. User logs out of Drone Package Delivery System.

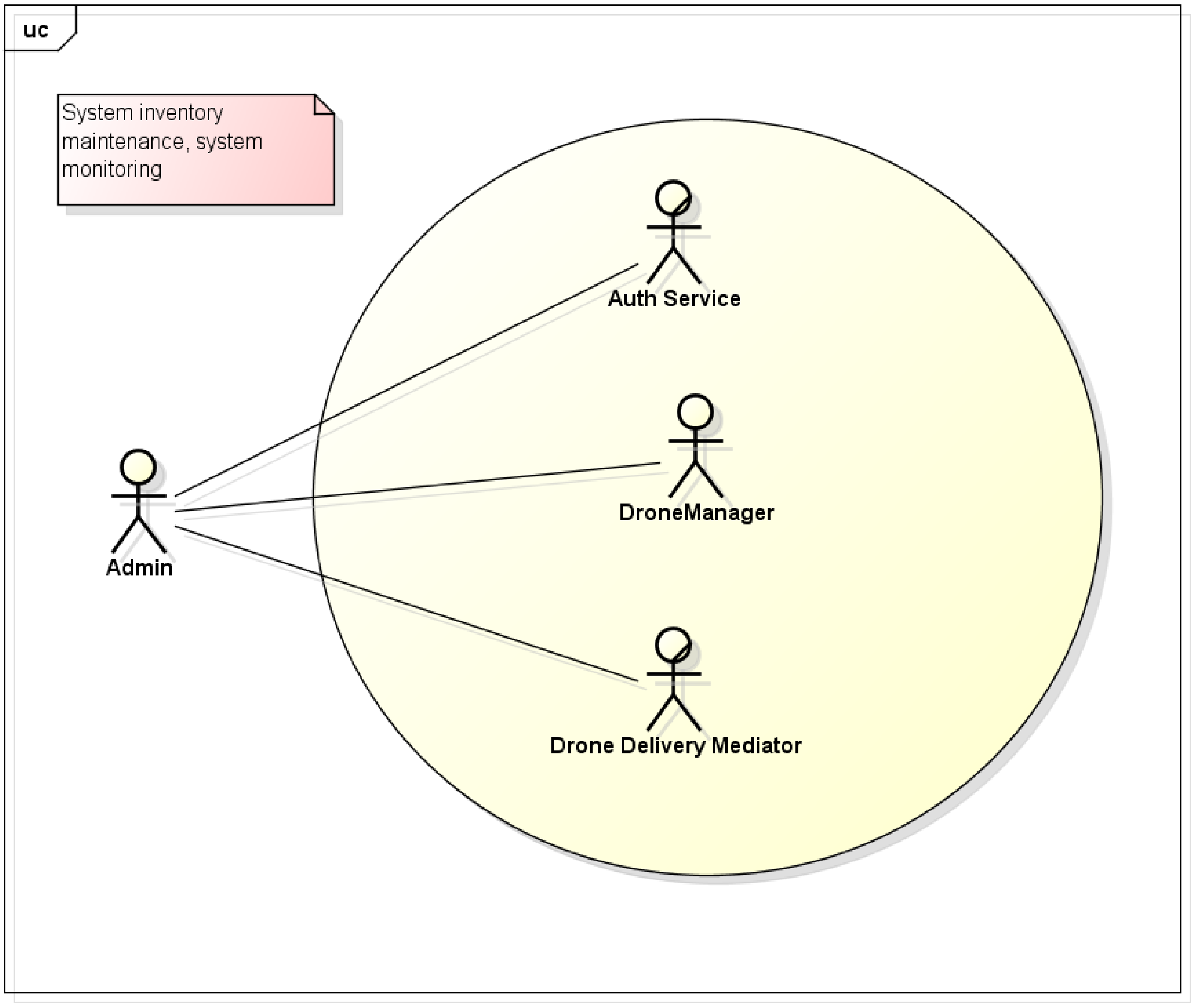
Delivery Creation (admin)
1. User phones or e-mails admin to request a delivery be created.
2. Admin logs into Drone Package Delivery System.
3. Admin creates delivery.
4. Admin logs out of Drone Package Delivery System.

Inventory/system maintenance, system monitoring
1. Admin logs onto Drone Package Delivery System.
2. Admins carries out functions.
3. Admin logs out of Drone Package Delivery System.

Use Case – Delivery Creation



Use Case – System Inventory Maintenance, System Monitoring



Implementation

The main components in the overall implementation are:

- The queue of scheduled deliveries in the DroneDeliveryMediator (scheduledDeliveries), sorted by the time the drone must leave to reach the pickup site in time (droneDepartureTime, in the class ScheduledDelivery, based on the Command pattern and which wraps the DeliveryManager.Delivery object that was initially created when the customer/service admin created a delivery.
- The queue of active, scheduled deliveries in the DroneDeliveryMediator sorted by expected delivery time (lateDeliveries).
- A messaging system whereby the DeliveryManager and DroneManager can each send messages to the DroneDeliveryMediator, and the DroneDeliveryMediator can send messages to either the DeliveryManager or DroneManager.
    - All messages essentially consist of a field indicating the service which sent the message, a status code indicating the type of message and an optional String descriptor.
    - In addition, messages sent between the DeliveryManager and DeliveryMediator also include a corresponding Delivery object, while messages sent between the DroneManager and DeliveryMediator also include a corresponding Drone object.

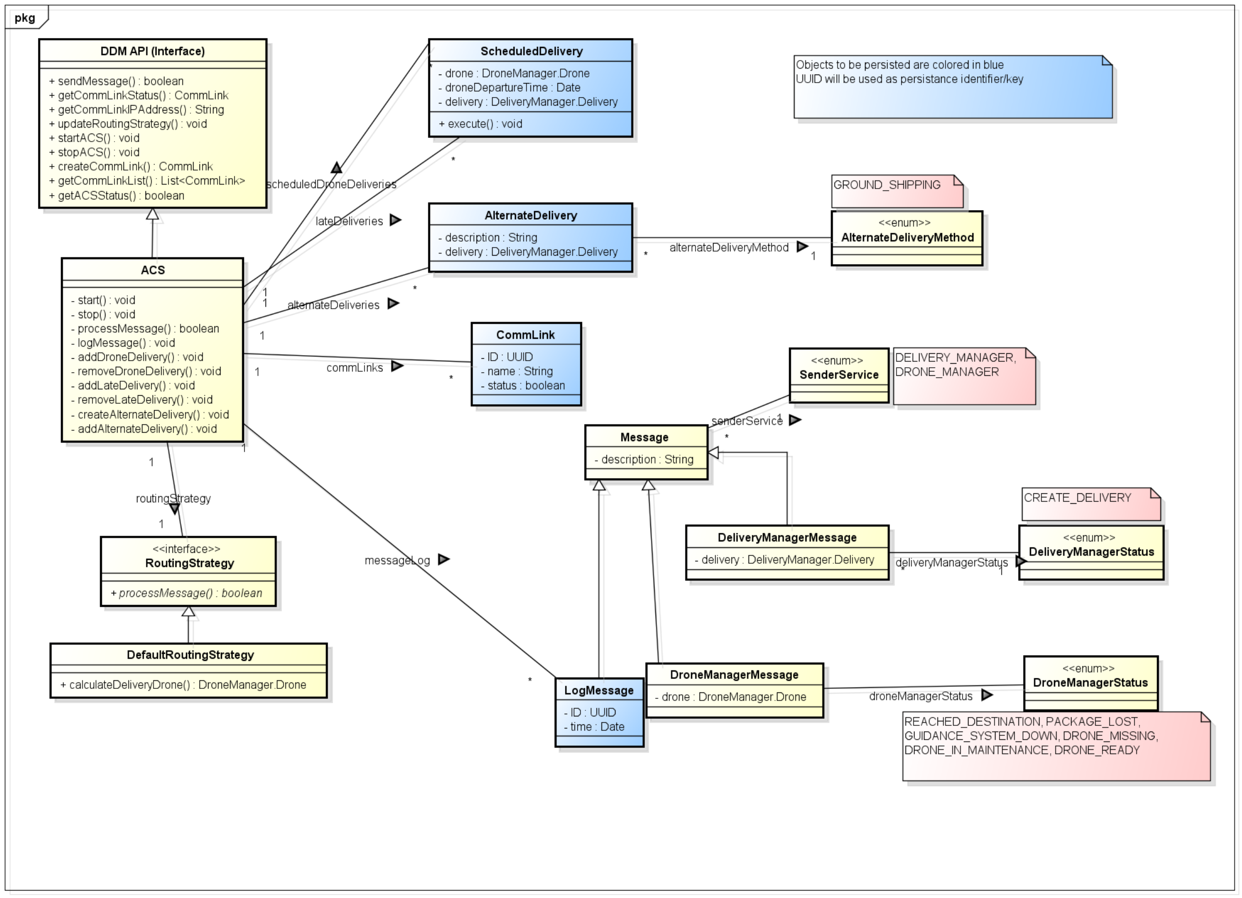
A given service can therefore make decisions just on the basis of state (this is especially true of the current RoutingStrategy implemented in the DeliveryMediator). The system has great flexibility since all that needs to be updated are the state/status enum classes in each given service and, in the case of the DeliveryMediator, the RoutingStrategy.

The following have been identified as changes in state that should initiate actions in the system (undoubtedly more will be determined in the future):

- DeliveryManager
    - Initial creation of an order.
- DroneDeliveryMediator
    - Time for for next droneDepartureTime in scheduledDeliveries is reached
    - Time for next expectedDeliveryTime in lateDeliveries is reached
- DroneManager
    - A drone is determined to be missing.
    - A drone is determined to require maintenance (needs to be fueled, repaired, etc.).
    - A drone being maintained (fueled, repaired) is now ready for operation.
- (Actual) Drone
    - Destination location reached
    - Package lost
    - Guidance system down
    - Guidance system up

In the case of messages sent by drones, the actual action that executed is also based on the current state of the drone as it appears in the DroneManagement system, which acts as a proxy for actual drones.

DroneDeliveryMediator Service Class Diagram


pkg
DDM API (Interface)
+ sendMessage() : boolean
+ getCommLinkStatus() : CommLink
+ getCommLinkIPAddress() : String
+ updateRoutingStrategy() : void
+ startACS() : void
+ stopACS() : void
+ createCommLink() : CommLink
+ getCommLinkList() : List<CommLink>
+ getACSStatus() : boolean
ScheduledDelivery
- drone : DroneManager.Drone
- droneDepartureTime : Date
- delivery : DeliveryManager.Delivery
+ execute() : void
Objects to be persisted are colored in blue
UUID will be used as persistance identifier/key
scheduledDroneDeliveries
lateDeliveries
GROUND_SHIPPING
AlternateDelivery
- description : String
- delivery : DeliveryManager.Delivery
<<enum>>
AlternateDeliveryMethod
alternateDeliveryMethod
ACS
- start() : void
- stop() : void
- processMessage() : boolean
- logMessage() : void
- addDroneDelivery() : void
- removeDroneDelivery() : void
- addLateDelivery() : void
- removeLateDelivery() : void
- createAlternateDelivery() : void
- addAlternateDelivery() : void
alternateDeliveries
CommLink
- ID : UUID
- name : String
- status : boolean
commLinks
<<enum>>
SenderService
DELIVERY_MANAGER,
DRONE_MANAGER
senderService
Message
- description : String
routingStrategy
CREATE_DELIVERY
<<interface>>
RoutingStrategy
+ processMessage() : boolean
DeliveryManagerMessage
- delivery : DeliveryManager.Delivery
deliveryManagerStatus
<<enum>>
DeliveryManagerStatus
messageLog
DefaultRoutingStrategy
+ calculateDeliveryDrone() : DroneManager.Drone
LogMessage
- ID : UUID
- time : Date
DroneManagerMessage
- drone : DroneManager.Drone
droneManagerStatus
<<enum>>
DroneManagerStatus
REACHED_DESTINATION, PACKAGE_LOST,
GUIDANCE_SYSTEM_DOWN, DRONE_MISSING,
DRONE_IN_MAINTENANCE, DRONE_READY
1
*

DroneDeliveryMediator Service Class Dictionary

API (Interface)
This interface class defines the DroneDeliveryMediator methods that will be exposed to external services that use the DroneDeliveryMediator. Its design is based on the Singleton, Façade and Mediator software design patterns.

Methods

Name	Signature	Description
sendMessage	(authToken:String, Message message):boolean	Public method to send a status/command message to the DroneDeliveryMediator. Returns true if the command was successfully executed, false otherwise
getCommlinkStatus	boolean (authToken:String, UUID commLinkID):boolean	Returns the operational status of the specified commLink.
getCommLinkIPAddress	(authToken:String, UUID commLinkID):String	Returns the IP address of the specified commLink.
updateRoutingStrategy	(authToken:String, RoutingStrategy newRoutingStrategy):void	Updates the RoutingStrategy used by the DroneDeliveryMediator ACS.
startACS	(authToken:String):void	Starts execution of the Runnable ACS object.
stopACS	(authToken:String):void	Stops execution of the Runnable ACS object.
createCommLink	(authToken:String, name:String, IPAddress:String):CommLink	Creates a ground commlink with the specified information and adds it to the commLinks map.
removeCommLink	(authToken:String, commlink:UUID):void	Removes the ground commlink with the specified ID from the commLinks map.
getCommLinkList	(authToken:String):List<CommLink>	Returns a list of all active commLinks.
getACSStatus()	(authToken:String):boolean	Returns the operational status of the ACS

ACS
Implements API and Java.Runnable. Its design in based on the Singleton and Mediator design patterns.

As specified in the requirements document, this class is the "brains" of the overall Drone Package Delivery System. It runs "forever" and executes actions appropriately when the current time matches the next time of the ScheduledDelivery object on the sorted scheduledDroneDeliveries List queue (sorted by droneDepartureTime) and/or when the current time matches the next time of the lateDeliveries List queue (also sorted by droneDepartureTime). It also executes actions appropriately in response to messages sent by external services.

Methods

Name	Signature	Description
start	():void	Private method to start execution of the Runnable ACS.
stop	():void	Private method to stop execution of the Runnable ACS.
processMessage	(authToken:String, message:Message):boolean	Private method to respond to a message sent by an external service. Called by the public API.processMessage() method, and calls routingStrategy.processMessage.
logMessage	(authToken:String, message:Message):void	Private method to log a message sent by an external service. Creates an appropriate LogMessage object and adds this to messageLog.
createScheduledDelivery	(authToken:String, DeliveryManager.Delivery delivery, DroneManager.Drone drone, Date	Private method to create a ScheduledDelivery object from the specified parameters.

	droneDepartureTime):ScheduledDelivery	
addDroneDelivery	(authToken:String, ScheduledDelivery delivery)	Private method to add a ScheduledDelivery object to the scheduledDroneDeliveries queue.
removeDroneDelivery	(authToken:String, ScheduledDelivery delivery)	Private method to remove the specified ScheduledDelivery object from the scheduledDroneDeliveries queue.
addLateDelivery	(authToken:String, ScheduledDelivery delivery)	Private method to add a ScheduledDelivery object to the lateDeliveries queue.
removeLateDelivery	(authToken:String, ScheduledDelivery delivery)	Private method to remove the specified ScheduledDelivery object from the lateDeliveries queue.
createAlternateDelivery	(authToken:String,)	
addAlternateDelivery	(authToken:String, AlternateDelivery delivery)	Private method to add an AlternateDelivery object to the alternateDeliveries queue.

Associations

Association Name	Type	Description
scheduledDroneDeliveries	List<ScheduledDelivery>	List of drone deliveries scheduled, sorted by droneDepartureTime.
lateDeliveries	List<ScheduledDelivery>	List of late deliveries scheduled, sorted by DeliveryManger.Delivery.promisedDeliveryTime. This list is provided mainly to determine if a delivery has not been made because of a missing drone.
alternateDeliveries	List<AlternateDelivery>	List of deliveries that were unable to be assigned to drones and that have been scheduled to be delivered by other means (e.g. ground shipping).
messageLog	List<LogMessage>	Message log
commLinks	Map<UUID, CommLink>	Set of commLinks being maintained by the ACS.
routingStrategy	RoutingStrategy	Current RoutingStrategy being applied by the ACS.

ScheduledDelivery

This class is a wrapper of a DeliveryManager.Delivery object, and makes use of the Command design pattern to make it possible initiate the delivery (though this is only in the case where the assigned drone is docked in a hanger. If the assigned drone is to execute the delivery directly following its previous delivery, the new delivery will be initiated immediately when the drone reports it has completed its previous delivery).

In addition to the DeliveryManager.Delivery object(a delivery as specified in the requirements document), a ScheduledDelivery object includes the drone assigned to the delivery as well as a droneDepartureTime Date object that indicates the time at which the drone assigned to the delivery should leave the hanger in order to pick up the scheduled delivery on time. This droneDepartureTime is irrelevant if the drone assigned to the delivery is currently not at a hanger and is picking up the package directly after a previous delivery.

Methods

Name	Signature	Description
execute	():void	Method allowing the delivery to be executed when appropriate.

Properties

Property Name	Type	Description
drone	DroneManager.Drone	Drone assigned to the delivery.
droneDepartureTime	Date	Time Drone should leave hanger to pick up the

		associated delivery on time.

AlternateDelivery
This class is used to describe a DeliveryManager.Delivery object that was unable to be scheduled for drone delivery and needs to be sent by an alternate method (e.g. ground shipping).

Properties

Property Name	Type	Description
alternateDeliveryMethod	AlternateDeliveryMethod	Alternate.
descriptionlivery	String	Optional descriptor allowing for additional comments (actual carrier used, if any, etc.).

Associations

Association Name	Type	Description
delivery	DeliveryManager.Delivery	(See DeliveryManager service)

AlternateDeliveryMethod
This enum class provides codes to describe alternate shipping methods (currently GROUND_SHIPPING). This class makes use of the Flyweight design pattern.

CommLink
Class used to describe a ground-based communication link available for use by the ACS.

Properties

Property Name	Type	Description
ID	UUID	Unique identifier for the ground commlink.
name	String	Name assigned to the given ground commlink.
status	boolean	Operational status of the given ground commlink (true if operational, false otherwise).

RoutingStrategy
This interface class allows for definition of the routing strategy to be used by the ACS. This class is based on the Strategy design pattern. (Note: I considered adding the method I implemented in the DefaultStrategy implementation class below to this interface, but decided to opt for more leeway in this initial design implementation).

Methods

Name	Signature	Description
processMessage	(authToken:String, message:Message):boolean	Code to process the given command/status message. Returns true if the command was executed successfully, false otherwise.

DefaultRoutingStrategy
Default routing strategy defined for the ACS.

Methods

Name	Signature	Description
processMessage	(authToken:String, message:Message):boolean	Code to process the given command/status message. Returns true if the command was executed successfully, false otherwise.

calculateDeliveryDrone	(authToken:String, delivery:DeliveryManager.Delivery):DroneManager.Drone	Returns the drone to be used for the specified delivery, or null if no such drone can be found.

Message
This class is a general description of any message that will be sent to the DroneDeliveryMediator by an external service. It is extended below by DeliveryManagerMessage and DroneManagerMessage.

Properties

Property Name	Type	Description
senderService	SenderService	Service sending the message
description	String	Optional message description.

SenderService
This enum class contains descriptions of the external services that may send the DeliveryDroneMediator messages (currently DELIVERY_MANAGER and DRONE_MANAGER). This class makes use of the Flyweight pattern.

DeliveryManagerMessage
This class extends Message and allows for description of messages that will be sent by the DeliveryManager service, including a specific DeliveryManager.Delivery object.

Properties

Property Name	Type	Description
deliveryManagerStatus	DeliveryManagerStatus	Message status code.
delivery	DeliveryManager.Delivery	Relevant DeliveryManager.Delivery object.

DeliveryManagerStatus
This enum class contains descriptions of possible codes for messages from the DeliveryManager (currently CREATE_DELIVERY). This class makes use of the Flyweight pattern.

DroneManagerMessage
This class extends Message and allows for description of messages that will be sent by the DroneManager service, including a specific DroneManager.Drone object.

Properties

Property Name	Type	Description
droneManagerStatus	DroneManagerStatus	Message status code.
drone	DroneManager.Drone	Relevant DroneManager.Drone object.

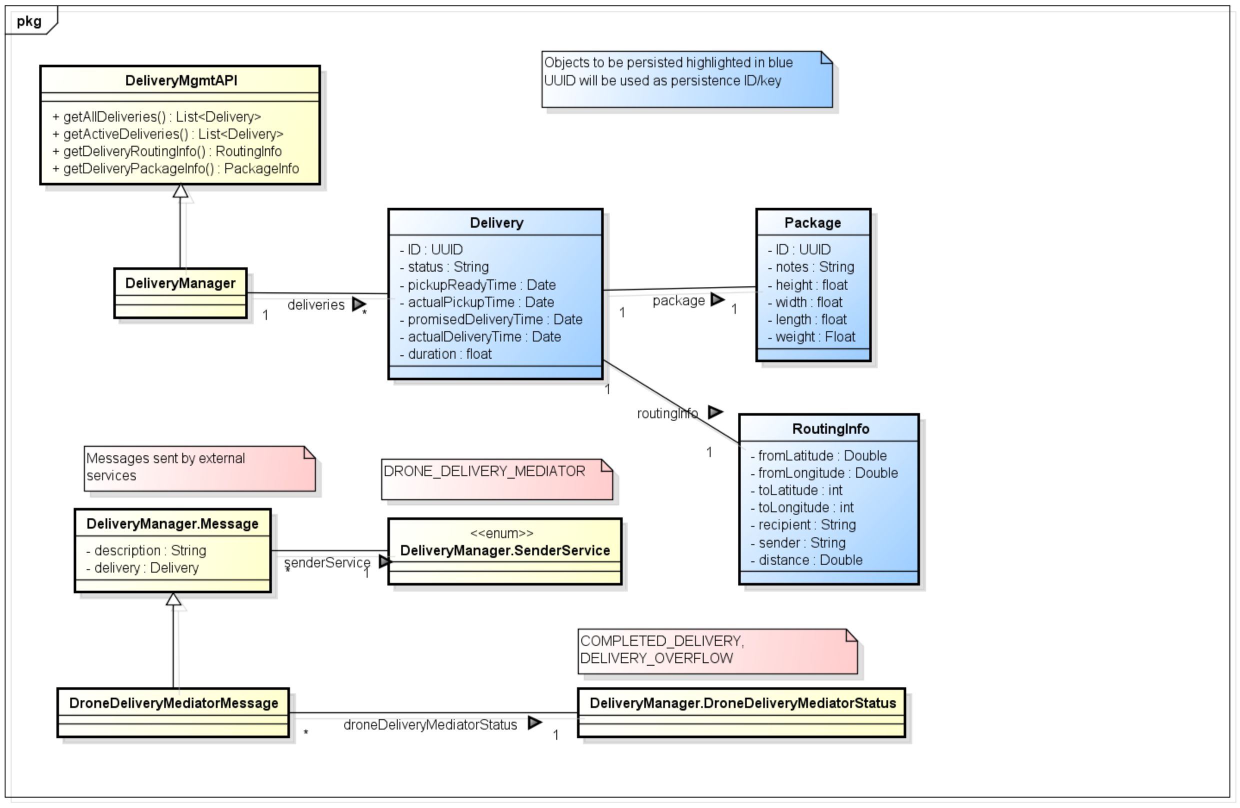
DroneManagerStatus
This enum class contains descriptions of possible codes for messages from the DroneManager (currently REACHED_DESTINATION, PACKAGE_LOST, GUIDANCE_SYSTEM_DOWN, GUIDANCE_SYSTEM_UP, DRONE_MISSING, DRONE_IN_MAINTENANCE, DRONE_READY). This class makes use of the Flyweight pattern.

LogMessage
This class extends Message and allows for creation of a status message that will be saved in the messageLog.

Properties

Property Name	**Type**	**Description**
ID	UUID	Unique identifier for Message object.
time	Date	Time message was saved.

DeliveryManager Service Class Diagram



DeliveryManager Service Class Dictionary

The DeliveryManager for the most part simply creates and maintains deliveries, and interacts very little with external services apart from notifying the DroneDeliveryMediator that a delivery has been created (so that service can schedule the delivery) and receiving notification from the DroneDeliveryMediator that a delivery has been completed (successfully or otherwise).

Assumptions: It is expected that any external user will be able to register with the Drone Package Delivery System and create a delivery, or that users may have Drone Package Delivery System admins create deliveries on their behalf. In addition, on the basis of the discussion board and articles regarding current technology, it is assumed that each delivery will only contain a single package.

Also, please note that a Delivery object in this Service is slightly different from the DroneDeliveryMediator ScheduledDelivery object which wraps it.

API

This class is based on the Singleton and Façade design pattern and exposes the DeliveryManager methods that external services may call.

Name	Signature	Description
getAllDeliveries	(authToken:String):List<Delivery>	Returns all deliveries which were created.
getActiveDeliveries	(authToken:String):List<Delivery>	Returns all deliveries which have not yet been completed.
getDelivery	(authToken:String, deliveryID:UUID):Delivery	Returns the Delivery object associated with the specified deliveryID
getDeliveryRoutingInfo	(authToken:String, deliveryID:UUID):RoutingInfo	Returns the RoutingInfo associated with the specified deliveryID
getDeliveryPackageInfo	(authToken:String, deliveryID:UUID):PackageInfo	Returns the PackageInfo associated with the specified deliveryID

DeliveryManager

Implements DeliveryManagementAPI and also provides additional private methods required to process deliveries. Based on the Singleton design pattern.

Delivery

This class is used to describe a Delivery.

Properties

Property Name	Type	Description
ID	UUID	Unique identifier assigned to the Delivery object.
pickupReadyTime	Date	Time the delivery is expected to be ready for pickup.
actualPickupTime	Date	Time the delivery was actually picked up.
promisedDeliveryTime	Date	Time

Message

This class is a general description of any message that will be sent to the DeliveryManager by an external service (this is currently only the DroneDeliveryMediator).

Properties

Property Name	Type	Description
senderService	SenderService	Service sending the message
description	String	Optional message description.
delivery	Delivery	Corresponding Delivery object for message (if any).

SenderService

This enum class contains descriptions of the external services that may send the DeliveryDroneMediator messages (currently DRONE_DELIVERY_MEDIATOR). This class makes use of the Flyweight pattern.

DroneDeliveryMediatorMessage

This class extends Message and allows for description of messages that will be sent by the DroneDeliveryMediator service.

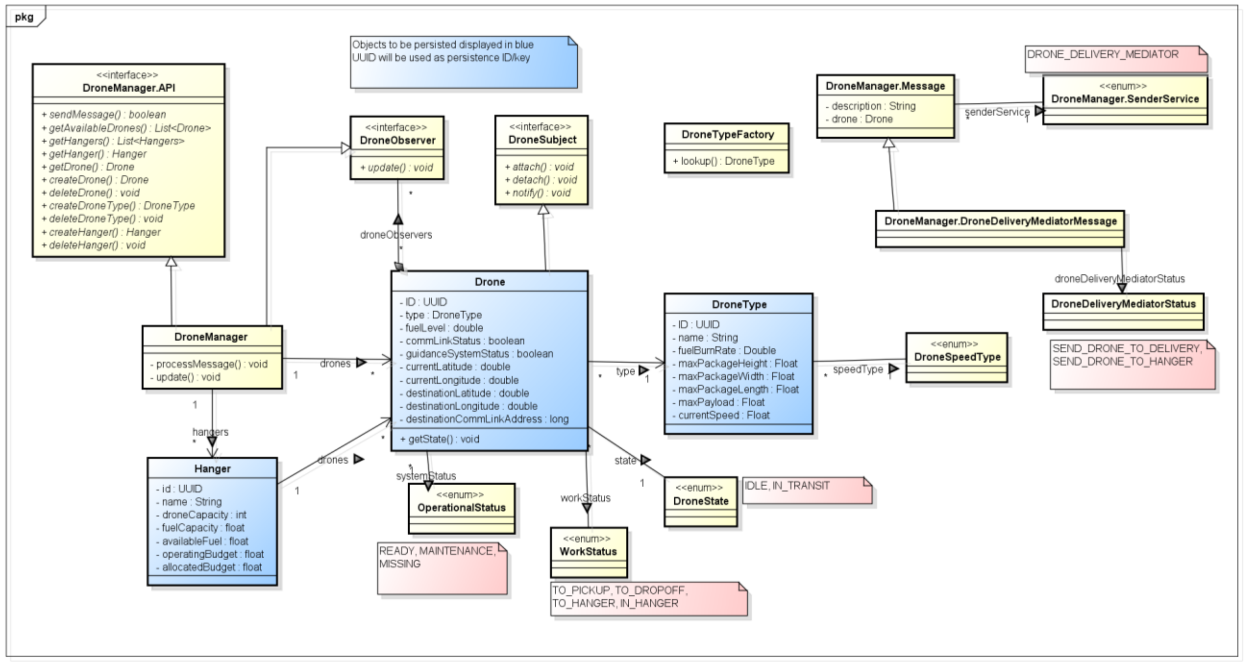
Properties

Property Name	Type	Description
droneDeliveryMediatorStatus	DroneDeliveryMediatorStatus	Message status code.

DroneDeliveryMediatorStatus

This enum class contains descriptions of possible codes for messages from the DroneDeliveryMediator (currently DELIVERY_OVERFLOW and COMPLETED_DELIVERY). This class makes use of the Flyweight pattern.

DroneManager Service Class Diagram



DroneManager Service Class Dictionary

The DroneManager Service is responsible for actual drone communication and for acutally updating the statuses of drones, whether or not these status updates are initiated by the DroneDeliveryMediator. The DroneManager also provides methods that make it possible to maintain inventory for drones and other related components, such as hangers.

The DroneManager makes use of the Observer design pattern for drone communication. The singleton DroneManager object attaches itself to observe active drones (i.e. those that are operationally ready and are on deliveries and not docked in hangers) then detaches itself when these drones are re-docked. DroneManager's private update() method, called by an active drone, takes into consideration the drone's operationalStatus as well as workStatus for proper execution.

The DroneManager also makes use of the Proxy design pattern (not properly implemented here, for reasons of time) to represent an actual drone, both to the DroneDeliveryMediator and to the DroneManager itself. The current design implementation assumes that the DroneManager itself contains most of the knowledge of a drone's work status and operational status, and that an actual drone only really knows the following:

- Its operational status (fuel level, commlink status, guidance system status)
- Current and destination locations (in latitude and longitude), and whether it is idle or in transit.
- The IP address of the ground commlink assigned to it.
- The following status changes:
    - It has reached its destination location
    - It has lost a package

As a result, the only messages initiated by an actual drone are:

- Destination location reached
- Package lost
- Guidance system down
- Guidance system up

and the DroneManager and DroneDeliveryMediator make decisions based on the additional status information on the drone maintained in the DroneManager.

API

This class defines the DroneManager methods that are to be exposed to external services. Makes use of the Singleton and Façade design patterns.

Methods

Name	Signature	Description
sendMessage	(authToken:String, Message message):boolean	Public method to send a status/command message to the DeliveryManager. Returns true if the command was successfully executed, false otherwise
getAvailableDrones	(authToken:String):List<Drone>	Returns a list of all DroneManager Drones that are operationally READY (regardless of whether they are currently on a delivery or not)
getAvailableDrones	(authToken:String, String droneType):List<Drone>	Returns a map of all DroneManager Drones of that are operationally READY and of the specified droneType.
getHangers	(authToken:String):List<Hanger>	Returns a list of of all Hangers currently being maintained in Drone Management system

getHanger	(authToken:String, UUID hangerID):Hanger	Returns the Hanger object with the specified ID (null if no such Hanger was found)
getDrone	(authToken:String, UUID droneID):Drone	Returns the Drone object with the specified ID (null if no such Drone was found)
createDrone	(authToken:String...):Drone	Creates a new Drone with the specified information and adds it to the drones Map.
deleteDrone	(authToken:String, droneID:UUID):void	Removes the specified Drone object from the drones Map.
createDroneType	(authToken:String...):DroneType	Creates a new DroneType with the specified information.
deleteDroneType	(authToken:String, droneID:UUID):void	Removes the specified DroneType object from DroneManager.
createHanger	(authToken:String...):Hanger	Creates a new Hanger with the specified information and adds it to the hangers Map.
deleteHanger	(authToken:String, droneID:UUID):void	Removes the specified Drone object from the hangers Map.

DroneManager

Implements the API and DroneObserver interfaces. Makes use of the Singleton and Observer design patterns.

Methods

Name	Signature	Description
processMessage	(Message message)	Private method
update	(drone:Drone, DroneStatus status):void	Private method to handle the notify() method called by a subject Drone (Observer pattern).

Associations

Association Name	Type	Description
drones	Map<UUID, Drone>	
hangers	Map<UUID, Hanger>	

Drone

Implements DroneSubject interface class. Provides methods/properties to describe a drone.

Methods

Name	Signature	Description
attach	void	Method to attach a DroneObserver object.
detach	void	Method to detach a DroneObserver object.
notify	(drone:Drone, message:DroneStatus):void	This method broadcasts a message to all currently attached DroneObserver objects, consisting of the drone itself and a status message.

Properties

Property Name	Type	Description
ID	UUID	Unique identifier for the Drone object.
type	DroneType	Drone's type.
fuelLevel	double	Drone's current level of fuel.
commLinkStatus	boolean	Operational status of the drone's commlink.
guidanceSystemStatus	boolean	Operational status of the drone's guidance system.
currentLatitude	double	Drone's current latitude.
currentLongitude	double	Drone's current longitude.
destinationLatitude	double	Latitude of the drone's destination.
destinationLongitude	double	Longitude of the drone's destination.

commLinkAddress	String	IP address of drone's commlink
groundCommLinkAddress	String	IP address of ground commlink assigned to drone.
state	DroneState	IN_TRANSIT or IDLE.

Associations

Association Name	Type	Description
droneObservers	List<DroneObserver>	Objects observing drone.

DroneSubject

Interface class making use of Observer design pattern. Used to define the methods that a drone that wishes to be observed must implement.

Methods

Name	Signature	Description
attach	void	Method to attach a DroneObserver object.
detach	void	Method to detach a DroneObserver object.
notify	void	Method to broadcast a message to all currently attached DroneObserver objects.

DroneObserver

Interface class making use of Observer design pattern. Used to define the methods that an object that wishes to observe a drone must implement.

Methods

Name	Signature	Description
update	void	Method describing action to take when a Drone object broadcasts a notification.

DroneState

This enum class defines the possible states an actual drone may be in (currently IN_TRANSIT or IDLE). Makes use of Flyweight design pattern.

DroneStatus

This enum class defines the possible status messages an actual drone may transmit (currently DESTINATION_REACHED, PACKAGE_LOST, GUIDANCE_SYSTEM_DOWN, GUIDANCE_SYSTEM_UP). Makes use of Flyweight design pattern.

DroneType

Class allowing for specification of a drone type (e.g. regular, SuperCarrier, etc.)

Properties

Property Name	Type	Description
name	String	Name of the drone type
fuelBurnRate	double	Fuel burn rate of drone type.
maxPackageHeight	float	Maximum package height in inches given drone type can support.
maxPackageLength	float	Maximum package length in inches given drone type can support.
maxPackageWidth	float	Maximum package width in inches given drone type can support.
maxPayload	float	Maximum payload weight, in pounds, given drone type

		can support.
speedType	DroneSpeedType	Speed type given drone type can support.

Hanger
Class allowing for description of a hanger used to store and maintain drones.

Properties

Property Name	Type	Description
ID	UUID	Unique identifier for hanger.
name	String	Name assigned to hanger.
droneCapacity	int	Maximum number of drones hanger can accommodate.
drones	List<Drone>	Drones currently in hanger.
fuelCapacity	float	Maximum amount of fuel hanger can store.
availableFuel	float	Current amount of fuel available in hanger.
operatingBudget	float	Operating budget for hanger.
allocatedBudget	float	Budget currently allocated.

OperationalStatus
enum class, making use of the Flyweight design pattern, used to describe possible drone operating statuses. Current values are READY, MISSING, MAINTENANCE

WorkStatus
enum class, making use of the Flyweight design pattern, used to describe possible drone delivery processing statuses. Current values are TO_PICKUP, TO_DROPOFF, TO_HANGER, IN_HANGER.

DroneSpeedType
enum class. Implements Flyweight design pattern. FAST, SLOW

DroneTypeFactory
Implements Factory, Flyweight design patterns. Used to generate/assign drone types.

Methods

Name	Signature	Description
lookup	(type:String):DroneType	Returns the DroneType with the specified identifier.
create	(name:String, fuelBurnRate:double, maxPackageHeight:float, maxPackageWidth:float, maxPackageLength:float, maxPayload:float):void	Creates a DroneType object with the specified parameters.

Message
This class is a general description of any message that will be sent to the DroneManager by an external service (this is currently only the DroneDeliveryMediator).

Properties

Property Name	Type	Description
senderService	SenderService	Service sending the message
description	String	Optional message description.
drone	Drone	Corresponding Drone object for message (if any).

SenderService

This enum class contains descriptions of the external services that may send the DroneManager messages (currently DRONE_DELIVERY_MEDIATOR). This class makes use of the Flyweight pattern.

DroneDeliveryMediatorMessage
This class extends Message and allows for description of messages that will be sent by the DroneDeliveryMediator service.

Properties

Property Name	Type	Description
droneDeliveryMediatorStatus	DroneDeliveryMediatorStatus	Message status code.

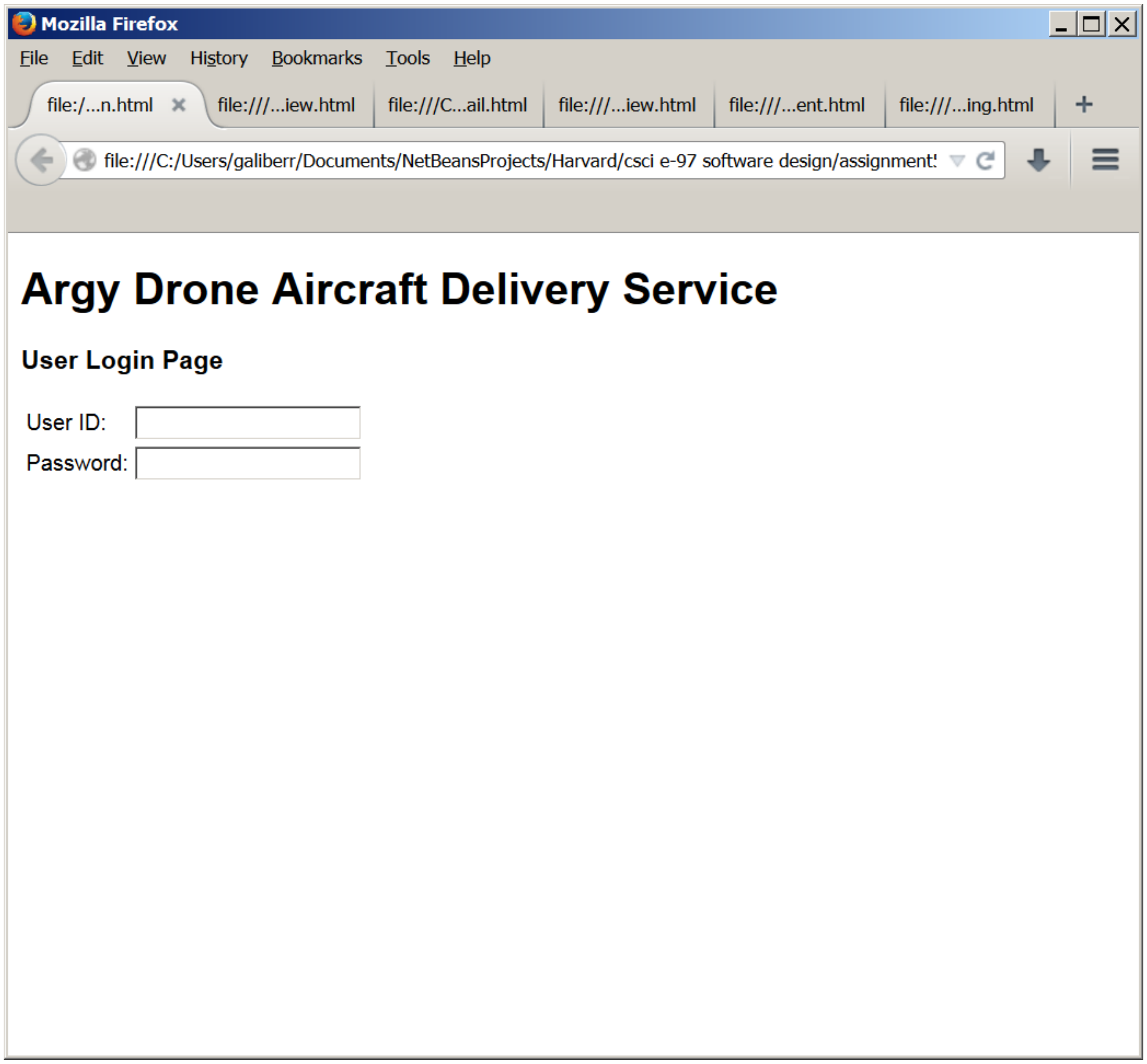
DroneDeliveryMediatorStatus
This enum class contains descriptions of possible codes for messages from the DroneDeliveryMediator (currently SEND_DRONE_TO_DELIVERY, SEND_DRONE_TO_HANGER). This class makes use of the Flyweight pattern.

Implementation Details

User Interfaces

The following user interfaces are intended mainly to identify the methods that will be required in the respective service APIs, and also provide a barebones view of what is expected on the user interfaces that will be provided for the system.

Login UI


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:/...n.html
file:///...iew.html
file:///C...ail.html
file:///...iew.html
file:///...ent.html
file:///...ing.html
file:///C:/Users/galiberr/Documents/NetBeansProjects/Harvard/csci e-97 software design/assignment!

Argy Drone Aircraft Delivery Service

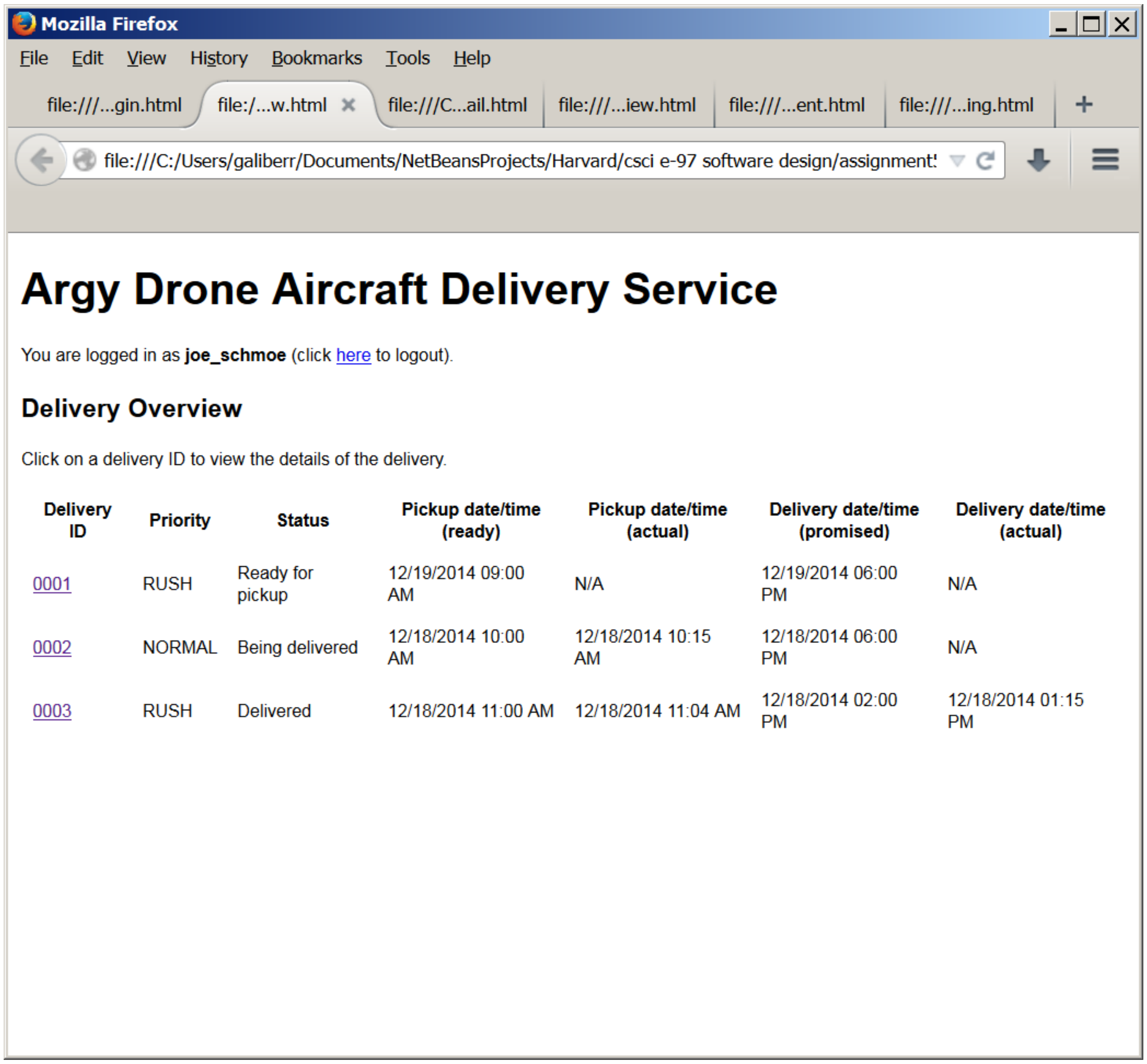
User Login Page

User ID:

Password:

<u>Methods required to support interface</u>:
AuthService – login()

Delivery Overview UI



Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///...gin.html
file:/...w.html
file:///C...ail.html
file:///...iew.html
file:///...ent.html
file:///...ing.html
file:///C:/Users/galiberr/Documents/NetBeansProjects/Harvard/csci e-97 software design/assignment!

Argy Drone Aircraft Delivery Service

You are logged in as **joe_schmoe** (click here to logout).

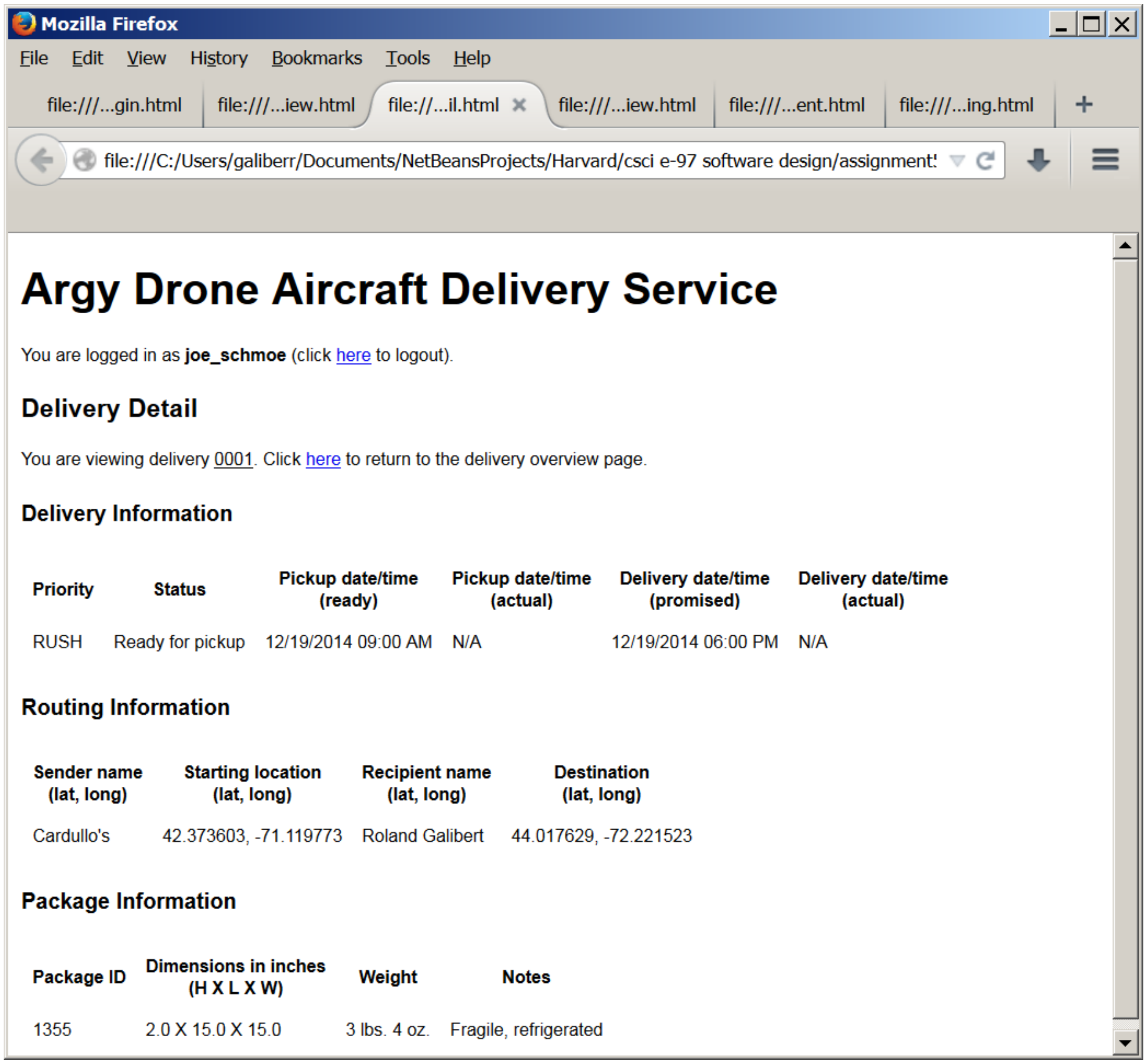
Delivery Overview

Click on a delivery ID to view the details of the delivery.

Delivery ID	Priority	Status	Pickup date/time (ready)	Pickup date/time (actual)	Delivery date/time (promised)	Delivery date/time (actual)
0001	RUSH	Ready for pickup	12/19/2014 09:00 AM	N/A	12/19/2014 06:00 PM	N/A
0002	NORMAL	Being delivered	12/18/2014 10:00 AM	12/18/2014 10:15 AM	12/18/2014 06:00 PM	N/A
0003	RUSH	Delivered	12/18/2014 11:00 AM	12/18/2014 11:04 AM	12/18/2014 02:00 PM	12/18/2014 01:15 PM

<u>Methods required to support interface</u>:
AuthService – logout()
DeliveryManager – getActiveDeliveries(), getDelivery(),

Delivery Detail UI


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///...gin.html
file:///...iew.html
file://...il.html
file:///...iew.html
file:///...ent.html
file:///...ing.html
file:///C:/Users/galiberr/Documents/NetBeansProjects/Harvard/csci e-97 software design/assignment!

Argy Drone Aircraft Delivery Service

You are logged in as **joe_schmoe** (click here to logout).

Delivery Detail

You are viewing delivery 0001. Click here to return to the delivery overview page.

Delivery Information

Priority	Status	Pickup date/time (ready)	Pickup date/time (actual)	Delivery date/time (promised)	Delivery date/time (actual)
RUSH	Ready for pickup	12/19/2014 09:00 AM	N/A	12/19/2014 06:00 PM	N/A

Routing Information

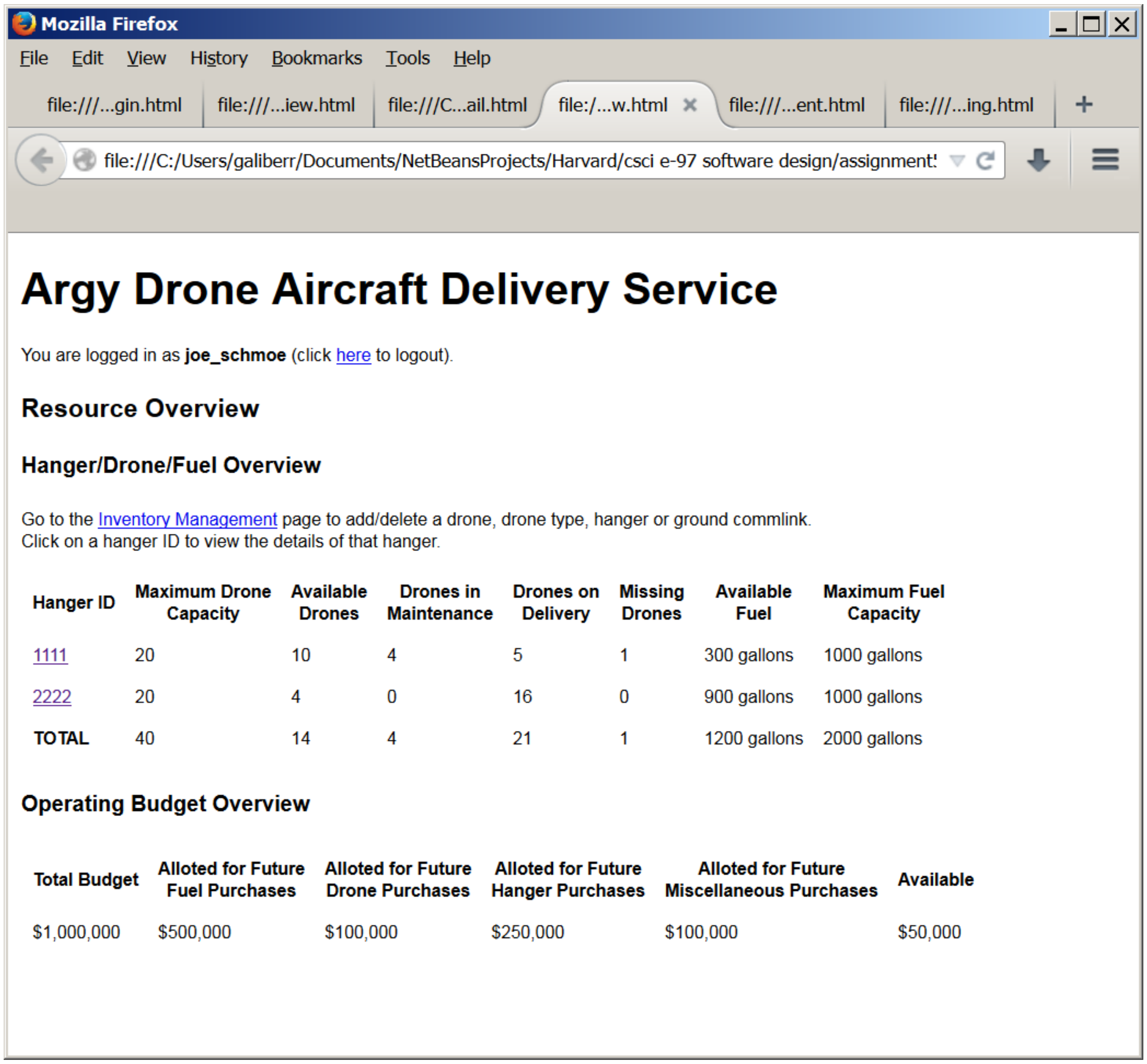
Sender name (lat, long)	Starting location (lat, long)	Recipient name (lat, long)	Destination (lat, long)
Cardullo's	42.373603, -71.119773	Roland Galibert	44.017629, -72.221523

Package Information

Package ID	Dimensions in inches (H X L X W)	Weight	Notes
1355	2.0 X 15.0 X 15.0	3 lbs. 4 oz.	Fragile, refrigerated

Methods required to support interface:
AuthService – logout()
DeliveryManager – getDelivery(), getDeliveryRoutingInfo(), getDeliveryPackageInfo()

Resource Overview UI


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///...gin.html
file:///...iew.html
file:///C...ail.html
file:/...w.html
file:///...ent.html
file:///...ing.html
file:///C:/Users/galiberr/Documents/NetBeansProjects/Harvard/csci e-97 software design/assignment!

Argy Drone Aircraft Delivery Service

You are logged in as **joe_schmoe** (click here to logout).

Resource Overview

Hanger/Drone/Fuel Overview

Go to the Inventory Management page to add/delete a drone, drone type, hanger or ground commlink.
Click on a hanger ID to view the details of that hanger.

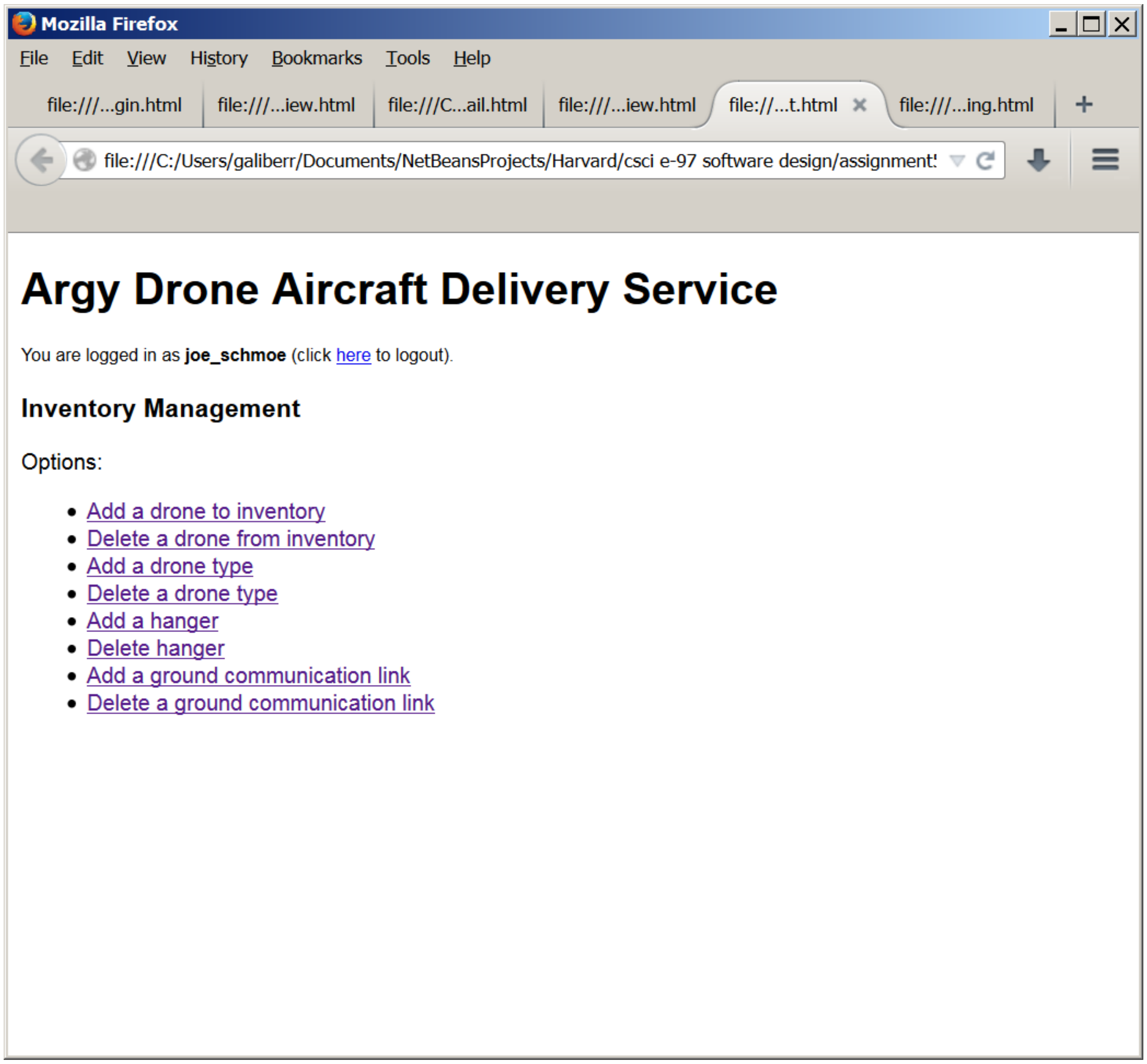
Hanger ID	Maximum Drone Capacity	Available Drones	Drones in Maintenance	Drones on Delivery	Missing Drones	Available Fuel	Maximum Fuel Capacity
1111	20	10	4	5	1	300 gallons	1000 gallons
2222	20	4	0	16	0	900 gallons	1000 gallons
TOTAL	40	14	4	21	1	1200 gallons	2000 gallons

Operating Budget Overview

Total Budget	Alloted for Future Fuel Purchases	Alloted for Future Drone Purchases	Alloted for Future Hanger Purchases	Alloted for Future Miscellaneous Purchases	Available
$1,000,000	$500,000	$100,000	$250,000	$100,000	$50,000

Methods required to support interface:
AuthService – logout()
DroneManger – getHanger()

Inventory Management UI


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///...gin.html
file:///...iew.html
file:///C...ail.html
file:///...iew.html
file://...t.html
file:///...ing.html
file:///C:/Users/galiberr/Documents/NetBeansProjects/Harvard/csci e-97 software design/assignment!

Argy Drone Aircraft Delivery Service

You are logged in as **joe_schmoe** (click here to logout).

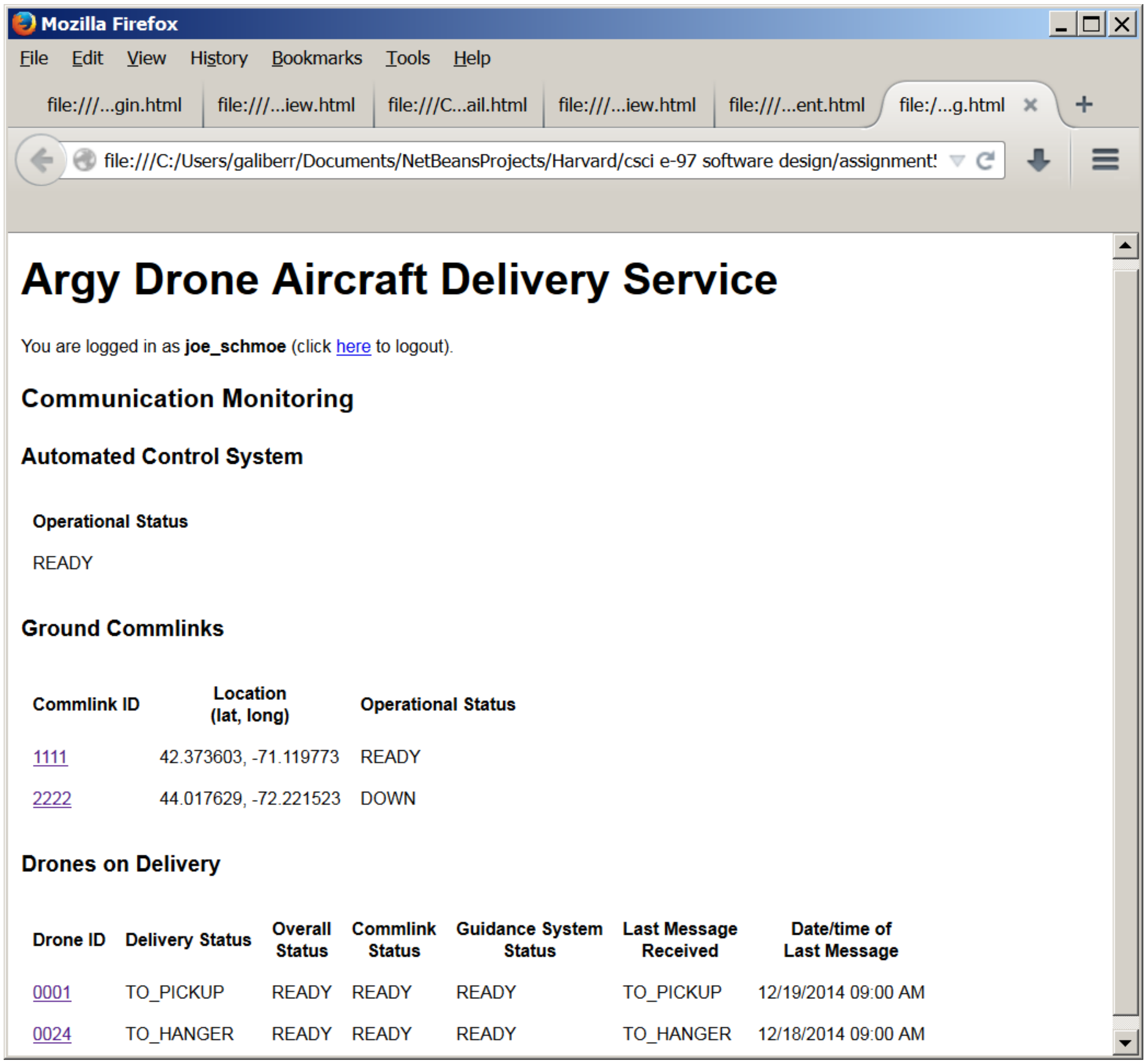
Inventory Management

Options:

- Add a drone to inventory
- Delete a drone from inventory
- Add a drone type
- Delete a drone type
- Add a hanger
- Delete hanger
- Add a ground communication link
- Delete a ground communication link

Methods required to support interface:
AuthService – logout()
DroneManager – createDrone(), deleteDrone(), createDroneType(), deleteDroneType(), createHanger(), deleteHanger()
DroneDeliveryMediator – createCommLink(), deleteCommLink()

Communication Monitoring UI


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///...gin.html
file:///...iew.html
file:///C...ail.html
file:///...iew.html
file:///...ent.html
file:/...g.html
file:///C:/Users/galiberr/Documents/NetBeansProjects/Harvard/csci e-97 software design/assignment!

Argy Drone Aircraft Delivery Service

You are logged in as **joe_schmoe** (click here to logout).

Communication Monitoring

Automated Control System

Operational Status
READY

Ground Commlinks

Commlink ID	Location (lat, long)	Operational Status
1111	42.373603, -71.119773	READY
2222	44.017629, -72.221523	DOWN

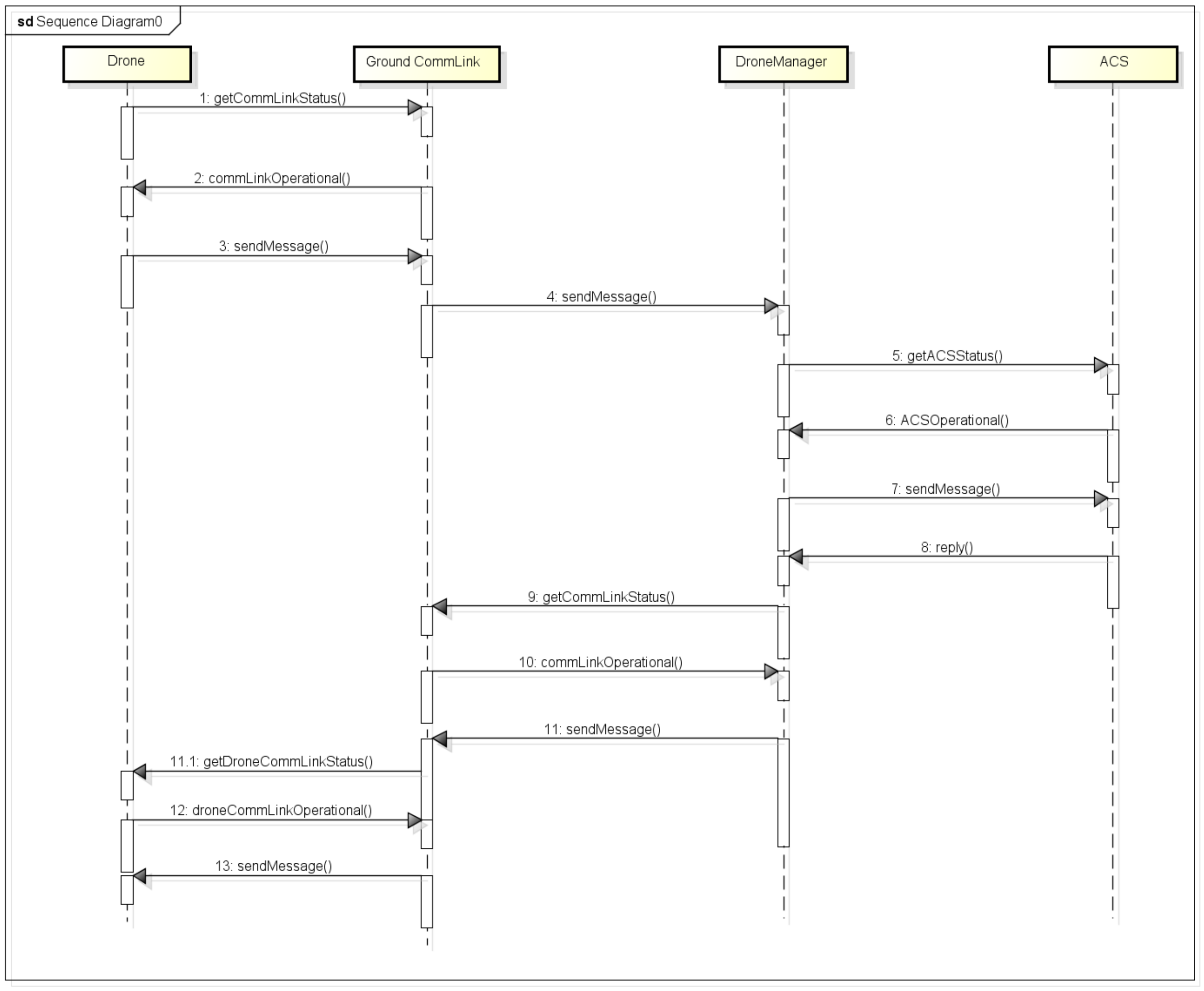
Drones on Delivery

Drone ID	Delivery Status	Overall Status	Commlink Status	Guidance System Status	Last Message Received	Date/time of Last Message
0001	TO_PICKUP	READY	READY	READY	TO_PICKUP	12/19/2014 09:00 AM
0024	TO_HANGER	READY	READY	READY	TO_HANGER	12/18/2014 09:00 AM

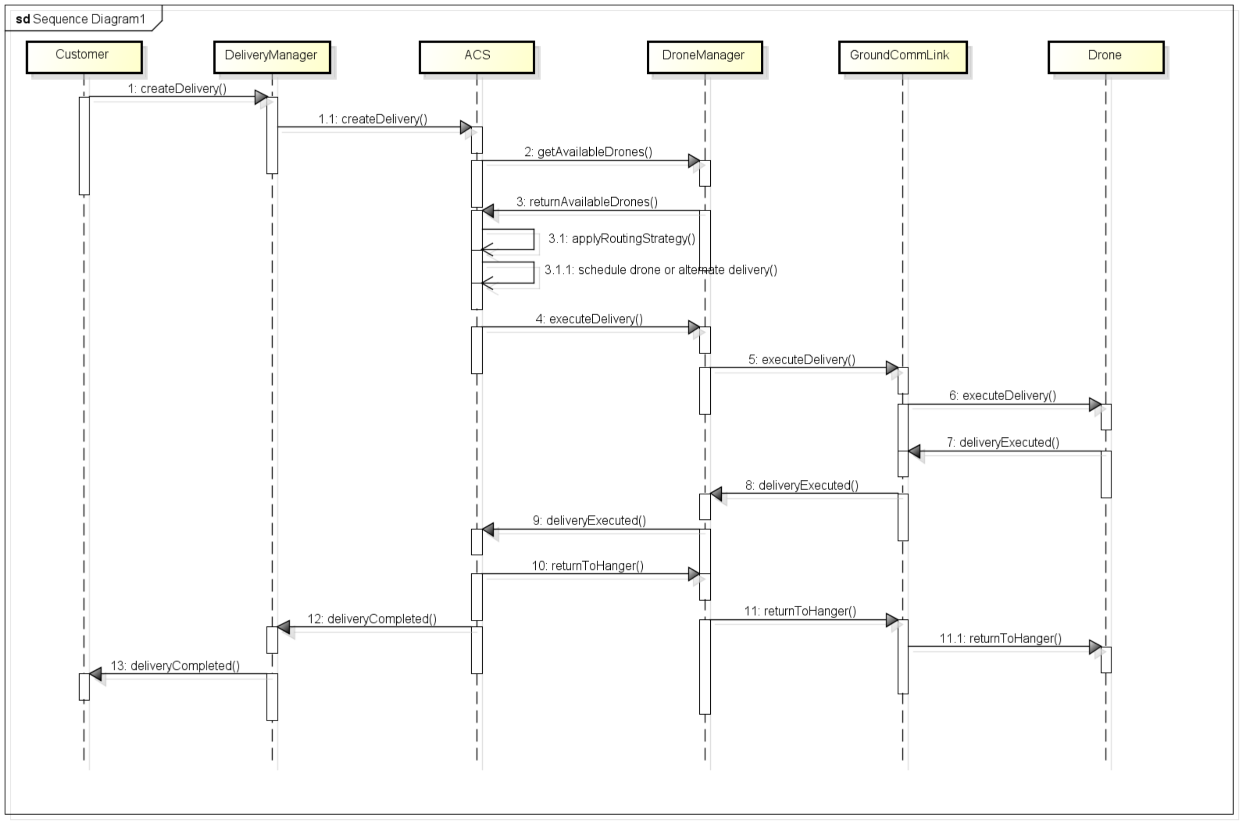
Methods required to support interface:
AuthService – logout()
DroneManager() – getActiveDrones()
DroneDeliveryMediator() – getCommLinkList(), getACSStatus()

Sequence diagrams

Message flow sequence displaying a status message sent by a drone

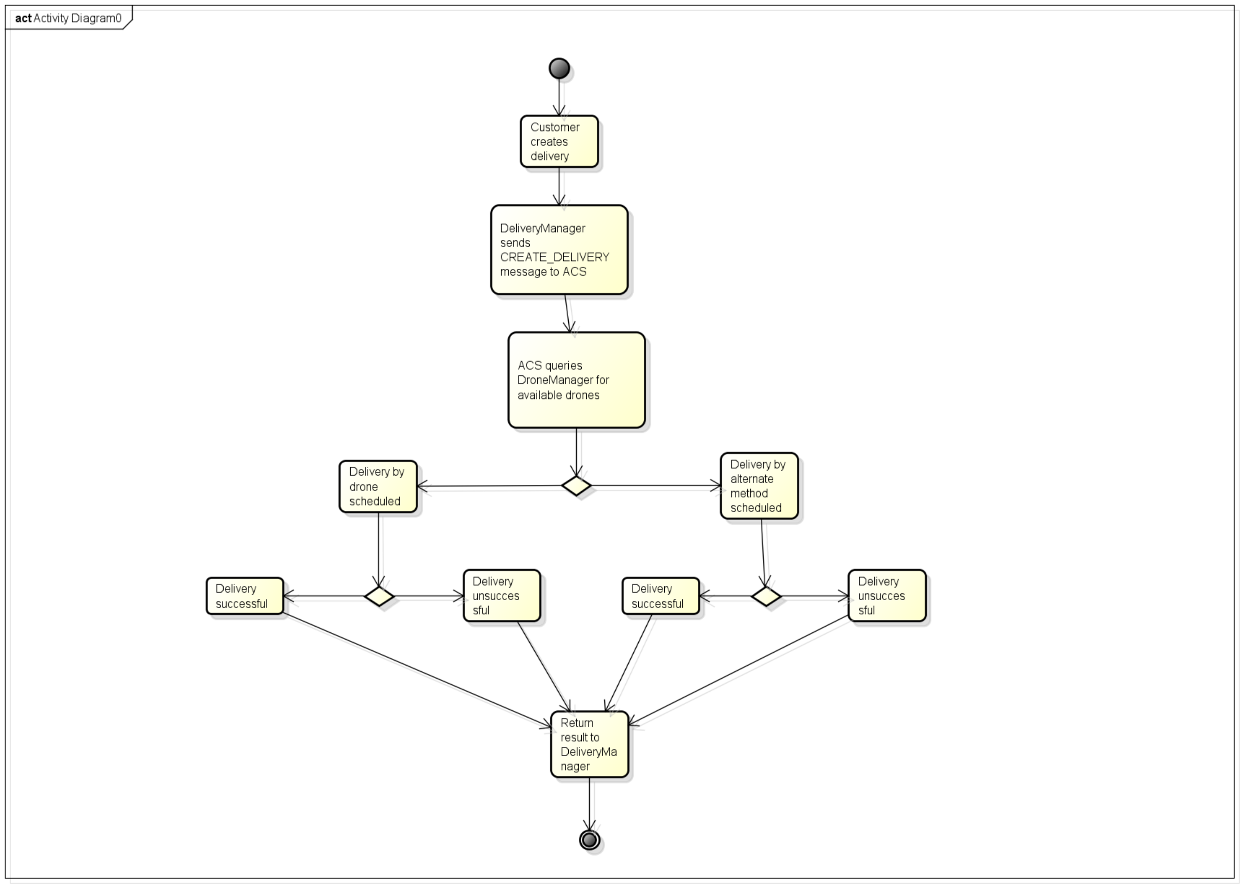


Message flow sequence for creating, scheduling and completing a delivery



Activity diagram

Activity diagram for creating, scheduling and completing a delivery


act Activity Diagram0
Customer creates delivery
DeliveryManager sends CREATE_DELIVERY message to ACS
ACS queries DroneManager for available drones
Delivery by drone scheduled
Delivery by alternate method scheduled
Delivery successful
Delivery unsuccessful
Delivery successful
Delivery unsuccessful
Return result to DeliveryManager

Testing

Separate modules should be created to test each of the DeliveryManager and DroneManager services and related user interfaces. This data can then be used to test the interactions/messaging executed by the DroneDeliveryMediator. The DroneDeliveryMediator test should also include testing of the RoutingStrategy process, ideally with multiple routing strategies. The DefaultRoutingStrategy should also be tested to ensure it works correctly.

Risks

Most of the risks involved in this system stem from the actual equipment used and not from the software system. There are a number of points of failure in the hardware including the multiple systems on each drone, the ground commlinks, and the connection to the automated control system. This is especially true since both the technology as well as processes (such as assigning drone deliveries which cannot be scheduled to conventional delivery methods) are very new.

There is a possible risk of lack of memory if the number of status messages to be saved turns out to be too high. The amount of delivery information to be saved appears to be manageable, especially when looking at this issue in conventional shipping companies. Saving inventory information (drones and hangers) does not appear to present a problem in terms of memory.

A final risk might be found in the routing strategy. An expert in scheduling should naturally be hired to develop a good routing strategy, or even one with no flaws, it would be too easy for a novice to create a very inefficient routing strategy or even one that doesn't work at all.