Tree Top Airlines Release 1

Design Documentation
Prepared by TEAM 6:

● Ian Randman	ixr5487@g.rit.edu
● Jonathon Chierchio	jpc8671@rit.edu
● Amber Harding	aah4320@rit.edu
● Matthew Antantis	mxa1051@rit.edu
● Mark Vittozzi	mev5063@rit.edu

Summary

Our team is responsible for the design and implementation of the Airline Flight Reservation Server. This system provides flight information for travelers using Treetop Airlines, and allows passengers to make, store, and delete flight reservations. It is a server-side system that provides an API used by client-side interfaces that Treetop Airlines passengers will use.

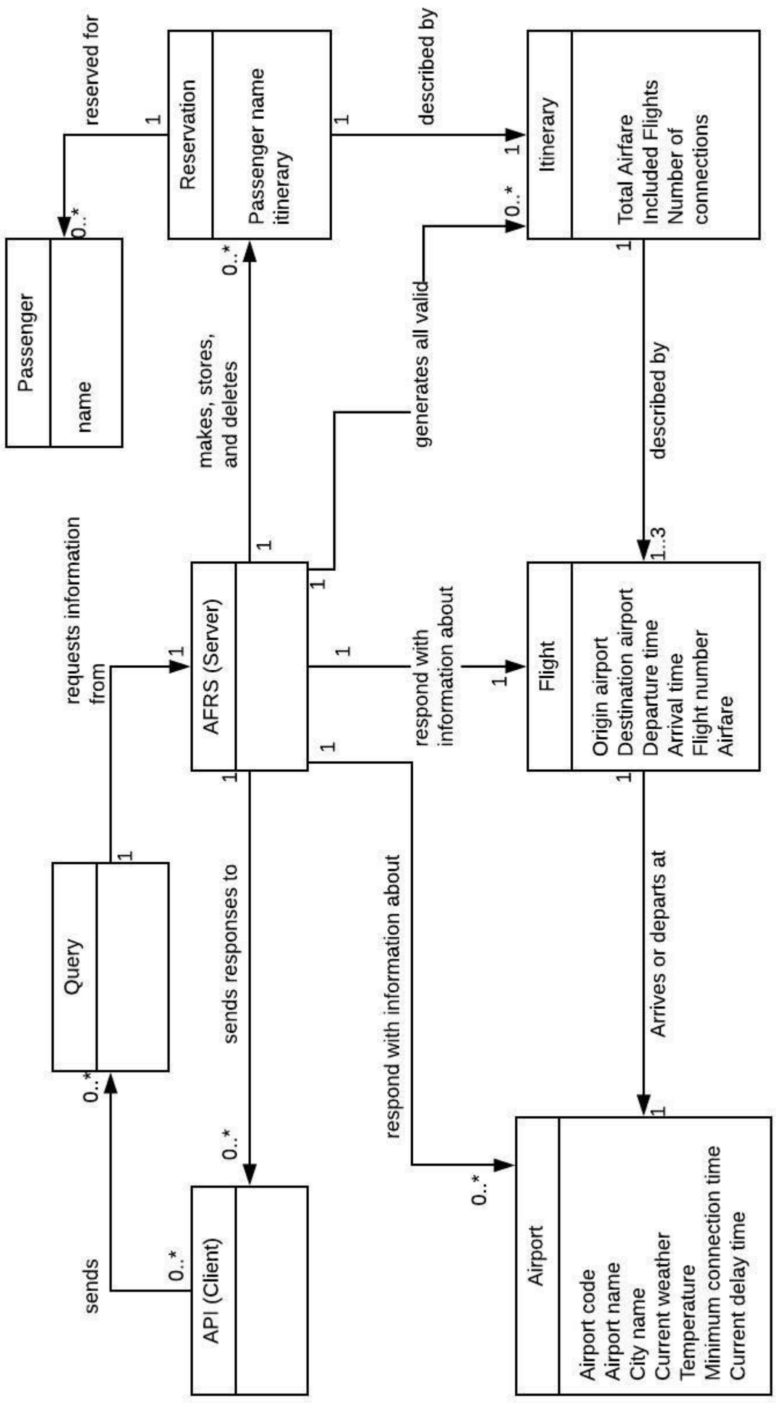
The AFRS is a text-based system where passengers requests to the system in the form of a text string and the system then forms a response to the client's request, also in the form of a text string. Every request the client makes is responded to by the system.

The AFRS will respond to queries for flight information. The system is required to store flight data for all flights between the cities in TTA's route network. Required flight data includes airfare, origin airport, destination airport, departure time, arrival time, and flight number. The system responds with all itineraries between the origin airport and destination airport, and itineraries may consist of flight combinations with multiple legs or direct flights. The client may request what order the flights will be listed in; either by departure time, arrival time, or airfare. The client may also limit the number of connections in flight combinations, but may request no higher than 2 connections. The system will automatically calculate the sum of airfares for every connection. If there are no itineraries matching the query, the system's response should be an empty string.

The AFRS will respond to queries for airport information as well. The client must provide the airport code in order to query for information, and the system shall respond with the airport name, current weather, temperature, and airport delay time.

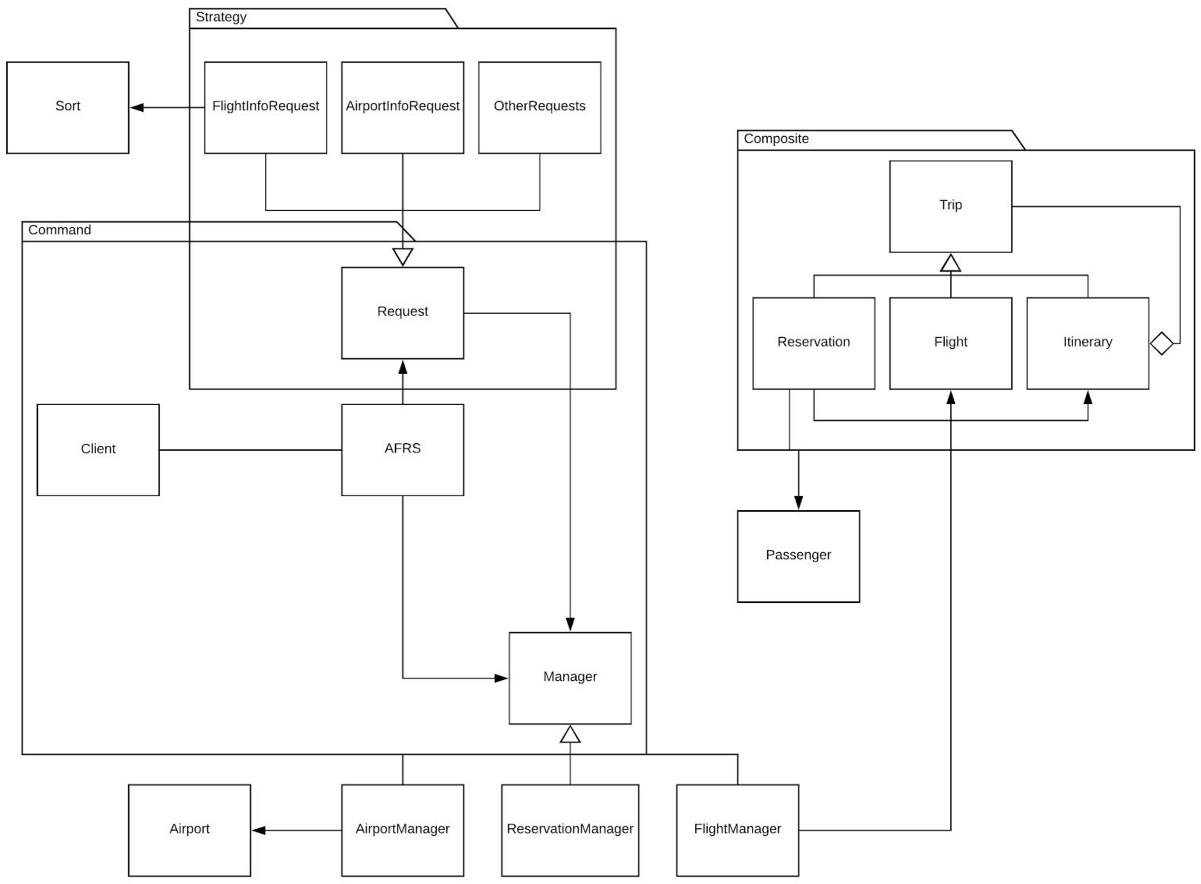
The AFRS will track reservations, and allow clients to create reservations for the most recent query for flight information. A request to make a reservation must have a passenger name, and the reservation will maintain the details of the reserved itinerary including total price and flight information for each flight in the itinerary. Clients an only have one reservation per origin-destination airport combination. Clients can also query for their reservations by using the passenger's name and optionally the origin or destination airports. Clients can also delete reservations. The system will persist reservations across system startups.

Domain Model

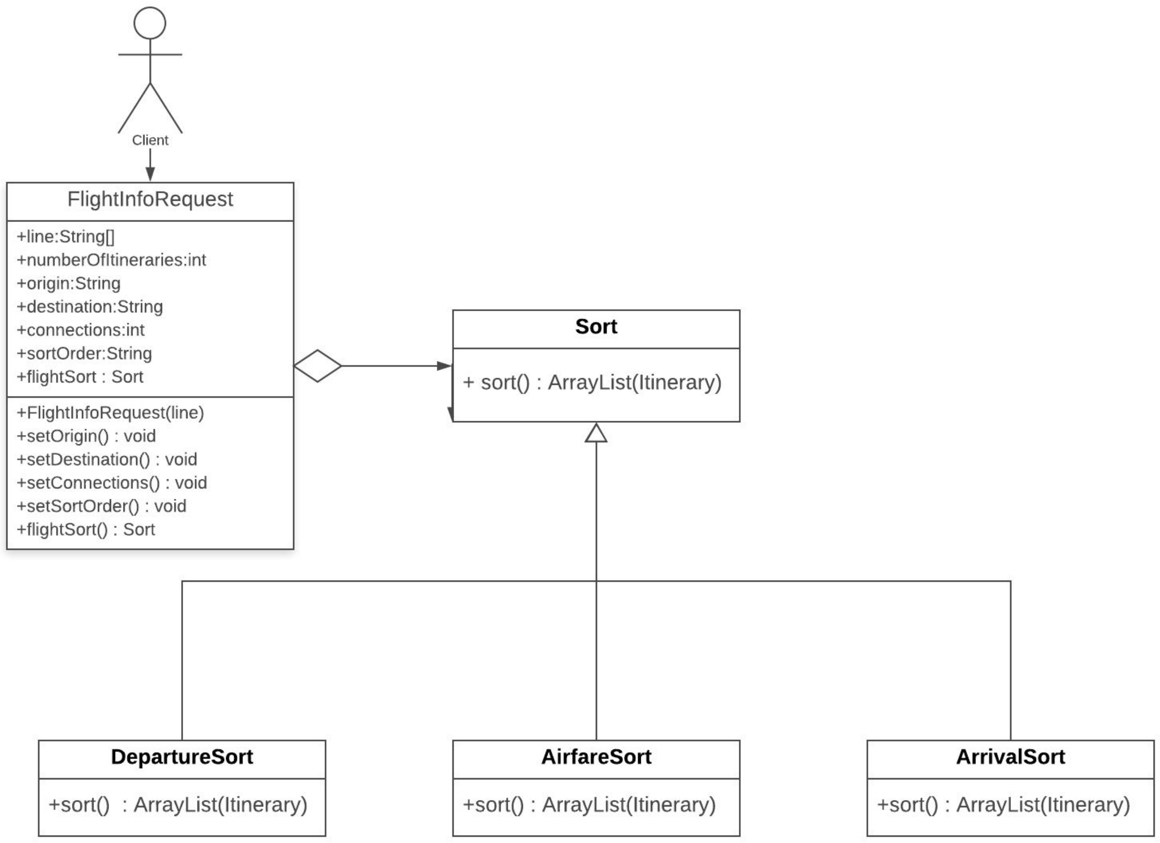


System Architecture

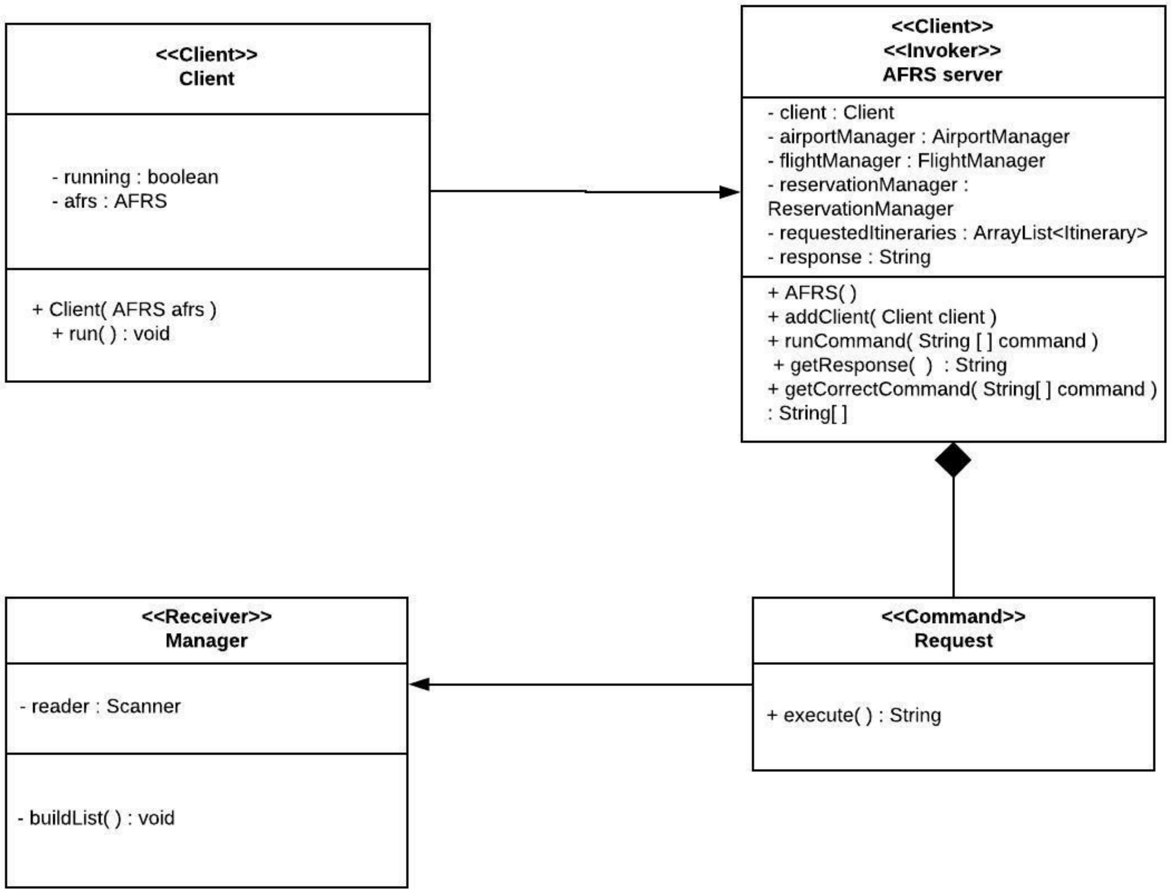
This section provides a model of the subsystem components that make up the overall software architecture for the project. Draw the subsystems as simple boxes with relationships between them. Provide a narrative that describes the responsibilities of each component and the interfaces that are provided between subsystems.



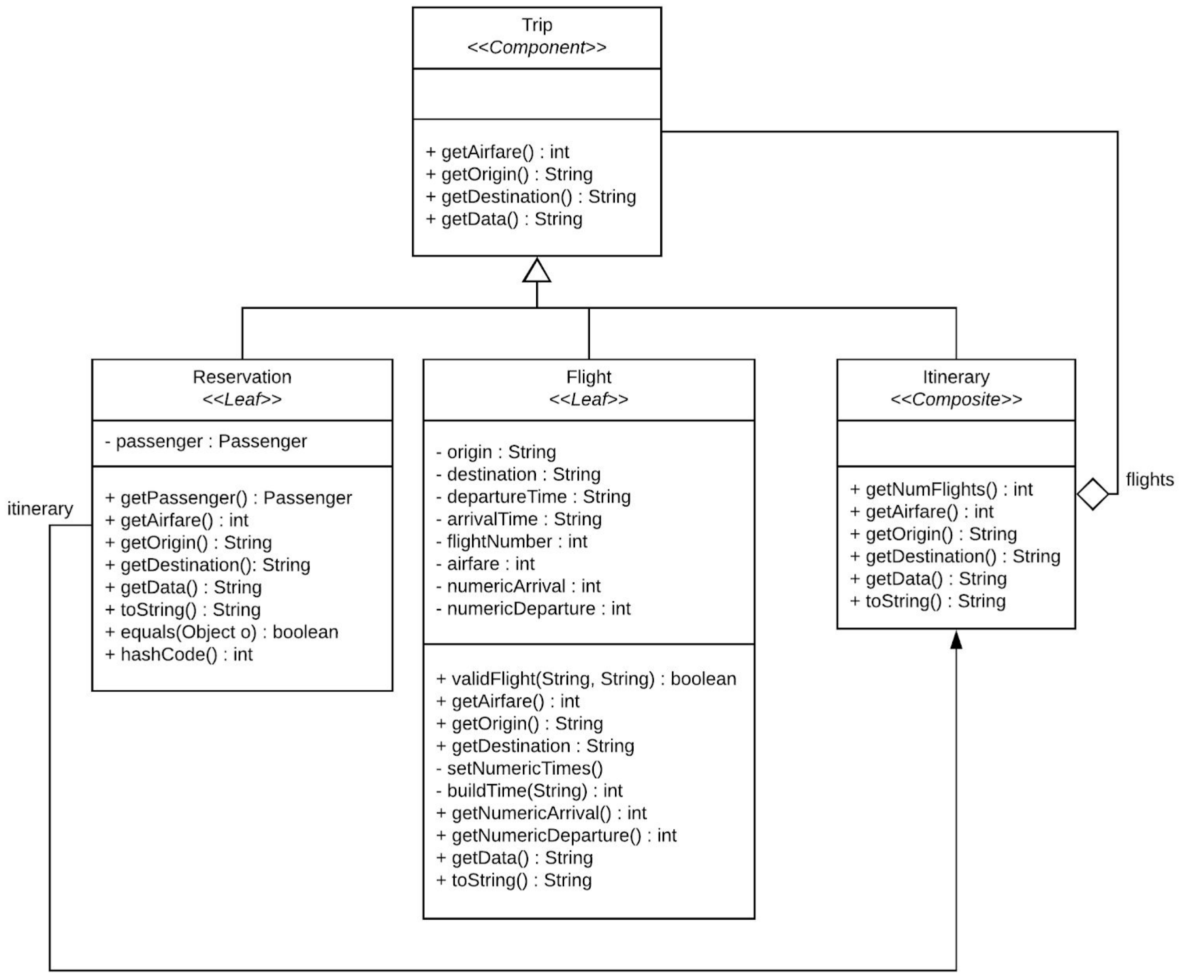
Strategy: A client makes a request for flight information. FlightInfoRequest uses Sort, an interface holding a generic sort method. DepartureSort, ArrivalSort, and AirfareSort use the sort method to return an ArrayList of Itineraries that gets sent back to the client in the form of a Response.



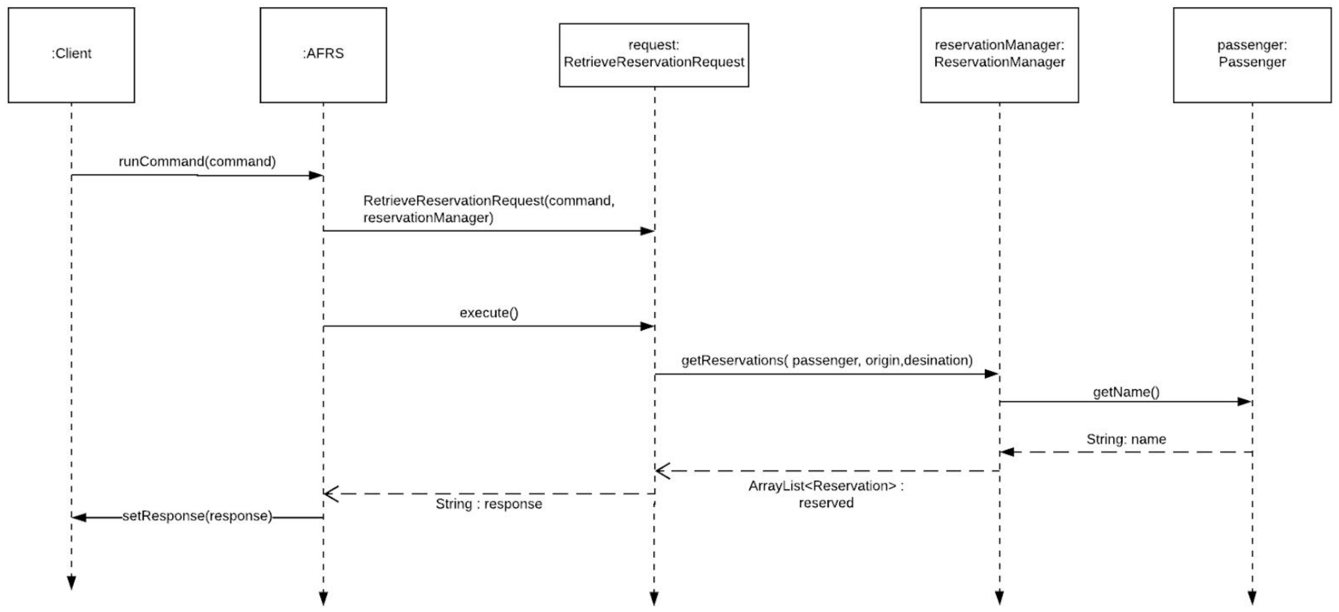
Command: The client receives user input and sends that to the server as an array of strings. The server behaves as a client and an invoker which is a deviation from the traditional command pattern. The server acts a client because it generates a request using the string array given to it by the client.The server also acts as the invoker because it calls the the execute method on the request. The request is the command and uses it’s execute method to generate a response. The manager acts as the receiver because it gets the information needed by the request so that the request is able to create a response. This is another deviation from the pattern because the request is performing the action of creating a response.


<<Client>>
Client
- running : boolean
- afrs : AFRS
+ Client(AFRS afrs)
+ run( ) : void
<<Client>>
<<Invoker>>
AFRS server
- client : Client
- airportManager : AirportManager
- flightManager : FlightManager
- reservationManager : ReservationManager
- requestedItineraries : ArrayList<Itinerary>
- response : String
+ AFRS( )
+ addClient(Client client)
+ runCommand(String [ ] command)
+ getResponse( ) : String
+ getCorrectCommand(String[ ] command) : String[ ]
<<Receiver>>
Manager
- reader : Scanner
- buildList( ) : void
<<Command>>
Request
+ execute( ) : String

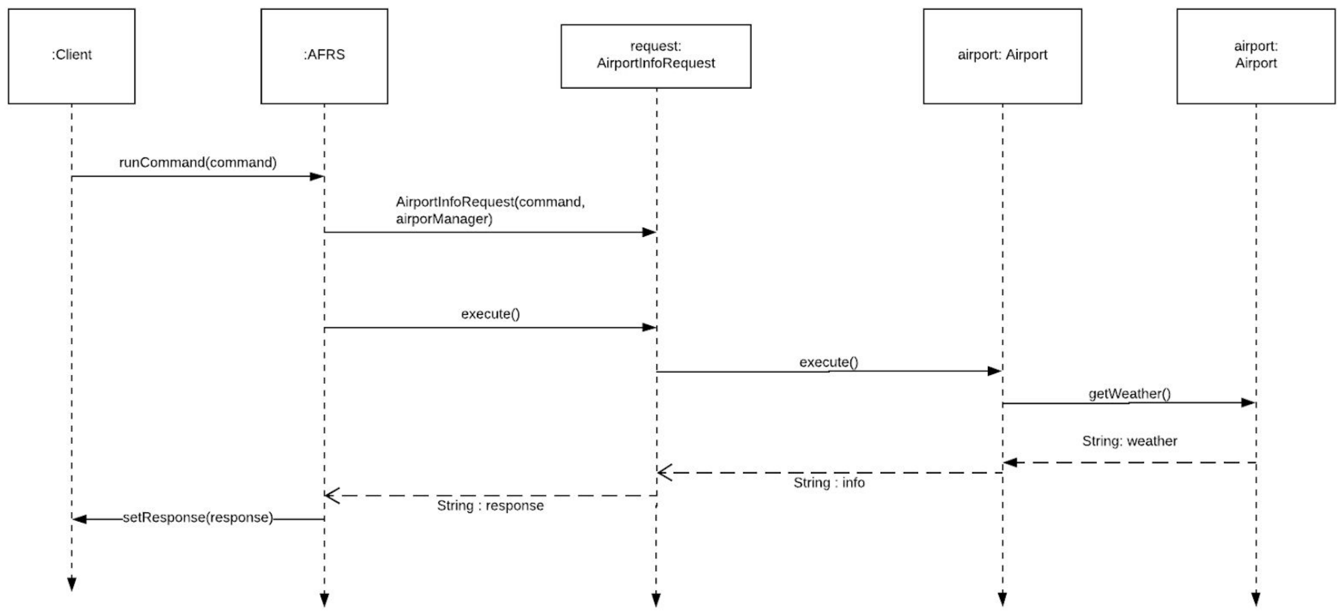
Composite: The system has a few different ways to describe from going on a trip from A to B. All instances of Trip have information about the origin, destination and cost. A Reservation has a Passenger and an Itinerary in addition to the standard Trip details. A Flight contains the information for a single flight from one airport to another. An Itinerary contains a list of flights that connect two airports.



Sequence Diagram for a client retrieving a reservation:


:Client
:AFRS
request: RetrieveReservationRequest
reservationManager: ReservationManager
passenger: Passenger
runCommand(command)
RetrieveReservationRequest(command, reservationManager)
execute()
getReservations(passenger, origin,desination)
getName()
String: name
ArrayList<Reservation> : reserved
String : response
setResponse(response)

Sequence Diagram for a client querying for weather information:



Subsystems

Client-Server Subsystem

Name: Client-Server Subsystem		**GoF pattern: Command**
Participants	AFRS (Server), Client, Request, Manager	
Class	**Role in GoF pattern**	**Participant's contribution in the context of the application**
AFRS (Server)	Client/Invoker	Server receives the text string input from the client and generates a request. In this way the server acts as a client. The server also acts as the invoker because it calls the the execute method on the request.
Client	Client	The client receives the user input and converts it into an array of strings which is then sent to the server.
Requests	Command	The request is generated as an object, which will be created by the server. The request is able to access the information that it needs to be able to complete it's execute method through the manager class.
Manager	Receiver	The manager gets the proper information for the request to be completed.
Deviations from the standard pattern: The Server will act as the client and the invoker, because the server creates a request object as well as calls the execute method on that request. The request is performing the action of creating a string response in its execute method, not the receiver in this case.		

Requirements being covered:
This covers the requirement of the system being able to create requests and generate responses that will be sent back to the client.
The API exchange shall be text-based.
The system shall receive requests from a client as text strings.
The system shall provide responses to client requests as text strings.

Itinerary Creation

Name: Itinerary creation		**GoF pattern: Composite**
Participants	Flight, Itinerary, Reservation	
Class	**Role in GoF pattern**	**Participant's contribution in the context of the application**
Flight	Leaf	Child of Trip
Itinerary	Composite	An itinerary is composed of multiple connecting flights.
Trip	Component	Trip is used so that a flight and itinerary may be treated the same way
Reservation	Leaf	Child of Trip

Deviations from the standard pattern:
We added a Trip class in order to make it so that all parts of the whole may be treated equally instead of having a linear hierarchy composed of flight -> itinerary -> reservation

Requirements being covered:
The AFRS shall track reservations.

Flight Sorting

Name: Flight Sorting		**GoF pattern: Strategy**
Participants FlightInfoRequest, Sort, DepartureSort, ArrivalSort, AirfareSort		
Class	**Role in GoF pattern**	**Participant's contribution in the context of the application**
FlightInfoRequest	Context	Users must create a FlightInfoRequest in order to choose a sort method
Sort	Strategy	Interface used
Departure Sort	ConcreteStrategyA	First strategy that sorts flight itineraries based on departure time
ArrivalSort	ConcreteStrategyB	Second strategy that sorts flight itineraries based on arrival time
AirfareSort	ConcreteStrategyC	Third strategy that sorts flight itineraries based on airfare
Deviations from the standard pattern:		
Requirements being covered: Sorting algorithm that sorts flight itineraries in different ways using abstraction instead of a conditional statement.		

Status of the Implementation

The product is in a completed state as all requirements are met and implemented to the best of our abilities.

Appendix

This section provides fine-grained design details for all of the classes in your design. You will capture this information using the CRC (Class-Responsibilities-Collaborators) card format below.

Class:	Main
Responsibilities:	Initializes the Client, Server, and starts the program.
Collaborators:	
Uses: Client, AFRS	**Used by:**
Author:	Matt Atlantis

Class:	Client
Responsibilities:	Handles the interactions with the user and communicates with the server.
Collaborators:	
Uses: AFRS	**Used by:** Main
Author:	Matt Atlantis

Class:	AFRS
Responsibilities:	Receives string array from user and uses that to create the proper request. It then calls execute on that request to get a response.
Collaborators:	
Uses: Request subclasses, Manager subclasses	**Used by:** Client
Authors:	Amber Harding, Mark Vittozzi, Matt Atlantis, Ian Randman

Class:	Request
Responsibilities:	Abstract class used as a base class for its subclasses.
Collaborators:	
Uses:	**Used by:** AirportInfoRequest, DeleteReservation, FlightInfoRequest, MakeReservation, PartialRequest, RetrieveReservation
Author:	Amber Harding, Mark Vittozzi

Class:	AirportInfoRequest
Responsibilities:	Contains execute method which gets airport information from the airport manager and creates a response as a string.
Collaborators:	
Uses: Request, AirportManager	**Used by:** AFRS
Author:	Amber Harding, Mark Vittozzi

Class:	DeleteReservationRequest
Responsibilities:	Contains execute method which calls reservationManager to remove the reservation requested, returns a string indicating whether this was successful or not.
Collaborators:	
Uses: Request, ReservationManager	**Used by:** AFRS
Author:	Amber Harding, Mark Vittozzi

Class:	FlightInfoRequest
Responsibilities:	Gets proper information from airport manager and flight manager to get correct data. Within the execute method this class also performs various error checking to ensure that the request can be completed.
Collaborators:	
Uses: Request, FlightManager	**Used by:** AFRS
Author:	Amber Harding, Mark Vittozzi, Ian Randman, Jon Chierchio, Matt Atantis

Class:	MakeReservationRequest
Responsibilities:	Contains execute method which calls the addReservation method on reservationManager, and returns a string indicating whether or not this call was successful
Collaborators:	
Users: Request, ReservationManagaer	**Used by:** AFRS
Author:	Amber Harding, Mark Vittozzi

Class:	RetrieveReservationRequest
Responsibilities:	Contains execute method which calls the getReservation method on reservationManager. Returns a string response.
Collaborators: ...	
Users: Request, ReservationManager	**Used by:** AFRS
Author:	Amber Harding, Mark Vittozzi

Class:	Sort
Responsibilities:	Interface used by sorting classes
Collaborators: ...	
Uses:	**Used by:** AirfareSort, ArrivalSort, DepartureSort, FlightInfoRequest
Author:	Amber Harding, Jon Chierchio

Class:	AirfareSort
Responsibilities:	Creates algorithm for sorting flight itineraries based on airfare
Collaborators:	
Uses: Sort	**Used by:** FlightInfoRequest
Author:	Jon Chierchio

Class:	ArrivalSort
Responsibilities:	Creates algorithm for sorting flight itineraries based on arrival time
Collaborators: ...	
Uses: Sort	**Used by:** FlightInfoRequest
Author:	Jon Chierchio

Class:	DepartureSort
Responsibilities:	Creates algorithm for sorting flight itineraries based on departure time
Collaborators: ...	
Uses: Sort	**Used by:** FlightInfoRequest
Author:	Jon Chierchio

Class:	Airport
Responsibilities:	Stores the information for an airport
Collaborators: ...	
Users:	**Used by:** AirportInfoRequest, AirportManager, AFRS
Author:	Matt Antantis

Class:	Flight
Responsibilities:	Stores the data for a specific flight
Collaborators: ...	
Users: Trip	**Used by:** FlightInfoRequest, FlightManager, ReservationManager, Itinerary, AFRS
Author:	Matt Antantis, Ian Randman

Class:	Itinerary
Responsibilities:	Stores the information for a series of flights between two airports
Collaborators: ...	
Users: Flight, Trip	**Used by:** AirportInfoRequest, FlightManager, ReservationManager, Reservation, AFRS
Author:	Matt Antantis, Ian Randman

Class:	Passenger
Responsibilities:	Stores the information for a Passenger for a reservation
Collaborators: ...	
Users:	**Used by:** Reservation, MakeReservationRequest, DeleteReservationRequest, RetrieveReservationRequest, ReservationManager, AFRS
Author:	Matt Antantis

Class:	Reservation
Responsibilities:	Stores the reservation for a passenger and an itinerary
Collaborators: ...	
Uses: Itinerary, Passenger	**Used by:** AFRS
Author:	Matt Antantis, Ian Randman

Class:	Trip
Responsibilities:	Component object for the composite pattern, the methods inside are overridden by its children
Collaborators: ...	
Uses:	**Used by:** Flight, Itinerary
Author:	Matt Antantis, Ian Randman

Class:	Manager
Responsibilities:	Abstract class used to organize the different managers
Collaborators: ...	
Uses:	**Used by:** AirportManager, FlightManager ReservationManager
Author:	Matthew Antantis

Class:	AirportManager
Responsibilities:	Manages information about all the airports in the System
Collaborators: ...	
Uses: Airport, Manager	**Used by:** AFRS
Author:	Matthew Antantis

Class:	FlightManager
Responsibilities:	Manages information about all the flights in the System
Collaborators: ...	
Uses: Flight, Manager	**Used by:** AFRS
Author:	Matthew Antantis

Class:	ReservationManager
Responsibilities:	Manages information about all the Reservations in the System
Collaborators:	
Uses: Manager, Flight, Itinerary, Passenger, Reservation	**Used by:** AFRS
Author:	Matthew Antantis