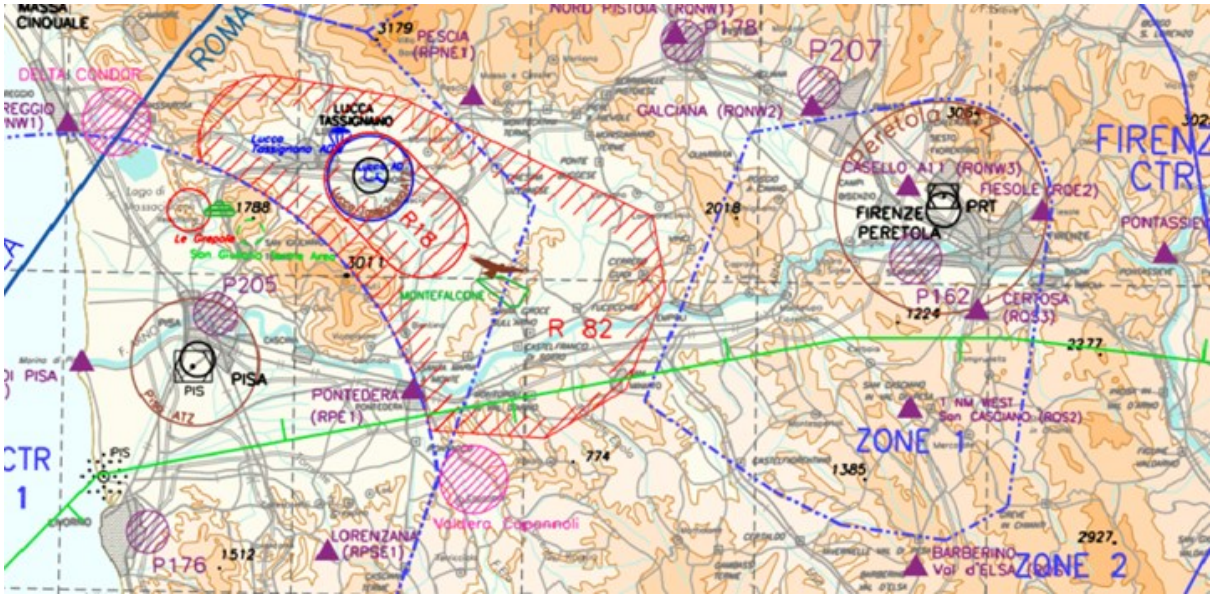
Niko Dalla Noce 6134764
Gabriele Vallario 5999189

Controllo Spazio Aereo

L’esigenza di creare dei “centri di assistenza al volo” che stabilissero le modalità d’uso di un determinato spazio aereo, nacque inizialmente sugli aeroporti. Furono gli aeroporti infatti, per la loro natura di “centri di confluenza” del traffico aereo, a far comprendere ai piloti la necessità di demandare ad un apposito “personale di terra”, la facoltà di stabilire le modalità d’uso dell’area di atterraggio e dello spazio aereo circostante. Difatti, quando sui più importanti aerodromi iniziarono a convergere più velivoli in arrivo da diverse direzioni nel medesimo arco orario, apparve chiara l’esigenza di cercare una soluzione per regolare tale traffico, al fine di prevenire le collisioni tra i velivoli in atterraggio, sorvoli e partenze. In questo caso, abbiamo implementato un sistema di allarme, che tramite un operazione esterna permette di capire se nello spazio aereo circostante vi sono “possibili” future collisione fra aeromobili che stimano uno stesso punto di riporto, in un arco temporale molto vicino ad una differenza di quota fra loro inferiore a 1000ft. Prima di iniziare un volo i piloti od i loro esercenti forniscono agli appropriati Enti del Servizio del Traffico Aereo il loro Piano di volo, documento nel quale è riportata la rotta dell’aeromobile. In questo caso il nostro spazio Aereo è costituito da una Regione Informazioni Volo (FIR), che racchiude al suo interno due Zone di Controllo (CTR) rispettivamente di Pisa e Firenze. Inoltre, ogni volo ha un’apparecchiatura di bordo chiamata Trasponder che ha un codice di 4 cifre che in caso di normalità è 7000. Lo stato di emergenza è associato al codice 7700. Durante la fase di crociera, se viene alterato lo stato del Trasponder, a seconda del codice, viene stampato su di un Monitor il problema riscontrato e nel caso sia un‘emergenza vengono fornite al pilota le informazioni per eseguire l’atterraggio nell’aeroporto più vicino ove sussistono le giuste condizioni meteo.



Soluzione Adottata

Abbiamo utilizzato il Pattern Observer in quanto quando ogni volo cambia il suo stato, altri oggetti devono essere informati ed aggiornarsi automaticamente. Ogni volo ha la sua lista di osservatori.
Nel caso dell'osservatore concreto Monitor, quando il volo cambia il suo stato, cioè cambia il suo codice Trasponder, a seconda del codice inserito, se rispetta una strategia, stampa in output un alert col codice inserito.

L'osservatore concreto EmergencyCall, ha il compito di effettuare la chiamata al numero passato al costruttore se e solo se il codice che è stato inserito è 7700.

L'osservatore EmergencyLanding, nel caso in cui il codice inserito è 7700, visto che si tratta di un'emergenza con necessità di atterraggio immediato, in base alla posizione dell'aeromobile, calcola l'aeroporto più vicino e se le condizioni meteo rispettano alcuni parametri, notifica al pilota dove atterrare e le coordinate dell'aeroporto.
In questo caso gli osservatori interrogano il soggetto e cercano di ricavare informazioni sufficienti per il loro aggiornamento o per portare al termine un loro compito.

Per realizzare lo spazio aereo, abbiamo creato una classe astratta Airspace che ha una lista di punti di Riporto. Ogni punto di Riporto, Report Point, potrà essere utilizzato dai piloti per essere aggiunto al loro piano di volo. Le classi concrete CTR e FIR estendono Airspace. Sono final perchè non possono essere estese.
La classe FIR è stata realizzata attraverso il pattern Singleton. E' una classe di cui esiste un solo esemplare, che costituisce uno strumento globale per tutti i client della classe.
Attraverso il costruttore privato, impedisco la creazione di oggetti della classe fuori dalla classe stessa. Attraverso un metodo statico che restituisce un riferimento a questo esemplare permetto ai client di avere un oggetto condiviso.

La classe ReportPoint ha due campi final che sono le coordinate del punto ed il nome dell'identificativo. Inoltre ha una lista di voli che hanno pianificato quel punto di riporto. Tutte le volte che un pilota aggiunge al suo piano di volo quel preciso punto di riporto, automaticamente il ReportPoint aggiunge il Flight alla sua lista.

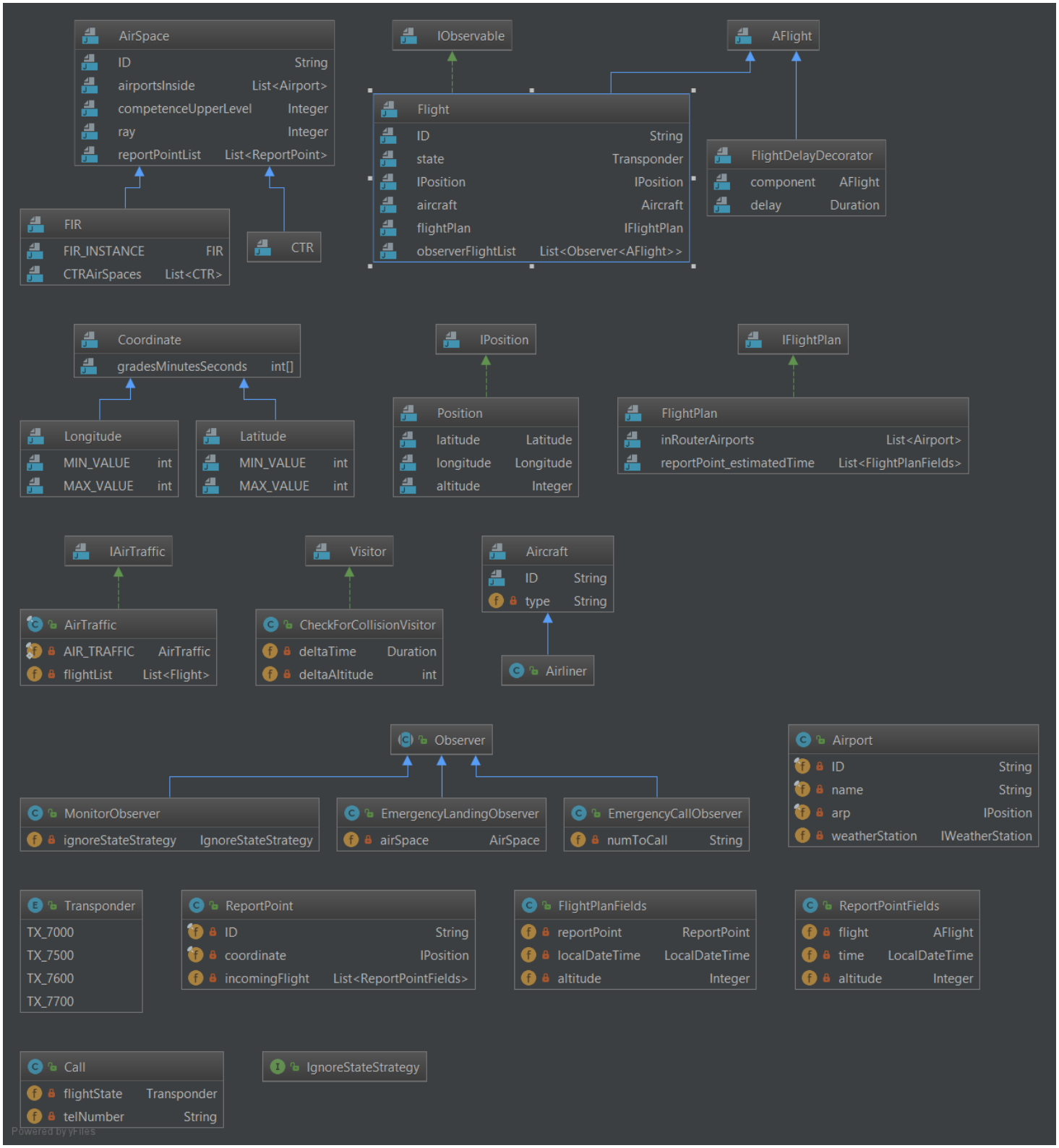
Per aggiungere un'operazione, senza modificare le classi esistenti abbiamo utilizzato il Pattern Visitor. L'operazione concreta in questione è la classe CheckForCollision, che implementa l'interfaccia Visitor. In questo caso la classe astratta Airspace, ha un metodo che accetta il Visitor.
L'operazione che viene eseguita in questo caso è quella di controllare se ci sono voli che hanno pianificato lo stesso punto di riporto, in un arco di tempo molto ravvicinato ad una quota fra loro di separazione inferiore a 1000 ft. Nel caso sussiste il rischio di collisione il Visitor stampa informazioni a riguardo del Punto di Riporto dove si verifica, indicando i voli coinvolti, gli orari e le loro rispettive quote.

Abbiamo utilizzato questo pattern sullo spazio aereo, poiché la struttura è stabile.

Attraverso il pattern Decorator abbiamo modificato il comportamento del volo a Run time, introducendo la singola decorazione FlightDelayDecorator. Abbiamo scelto di applicare questo pattern perchè specializza il comportamento del singolo oggetto in fase dinamica, rispetto all'ereditarietà che aggiunge comportamenti su tutta la classe di oggetti.

```
--------------------------------
Collisions@ Report 2
@22:07:56/ 23 dic 2017 +- min 5
   Flight Due Altitude: 1500
   Flight Uno Altitude: 2000
@23:07:56/ 23 dic 2017 +- min 5
   Flight Tre Altitude: 1400
   Flight Uno Altitude: 1000
--------------------------------
Collisions@ Report 3
@23:17:56/ 23 dic 2017 +- min 5
   Flight Uno Altitude: 1000
   Flight Tre Altitude: 1900

Land in Roma airport with coordinates Latitude[41, 53, 35] Longitude[12, 28, 58] alt 0
Flight Uno TX_7700
Emergency call: 054236984 with state TX_7700
```


AirSpace
ID String
airportsInside List<Airport>
competenceUpperLevel Integer
ray Integer
reportPointList List<ReportPoint>
IObservable
AFlight
Flight
ID String
state Transponder
IPosition IPosition
aircraft Aircraft
flightPlan IFlightPlan
observerFlightList List<Observer<AFlight>>
FlightDelayDecorator
component AFlight
delay Duration
FIR
FIR_INSTANCE FIR
CTRAirSpaces List<CTR>
CTR
Coordinate
gradesMinutesSeconds int[]
IPosition
IFlightPlan
Longitude
MIN_VALUE int
MAX_VALUE int
Latitude
MIN_VALUE int
MAX_VALUE int
Position
latitude Latitude
longitude Longitude
altitude Integer
FlightPlan
inRouterAirports List<Airport>
reportPoint_estimatedTime List<FlightPlanFields>
IAirTraffic
Visitor
Aircraft
ID String
type String
AirTraffic
AIR_TRAFFIC AirTraffic
flightList List<Flight>
CheckForCollisionVisitor
deltaTime Duration
deltaAltitude int
Airliner
Observer
Airport
ID String
name String
arp IPosition
weatherStation IWeatherStation
MonitorObserver
ignoreStateStrategy IgnoreStateStrategy
EmergencyLandingObserver
airSpace AirSpace
EmergencyCallObserver
numToCall String
Transponder
TX_7000
TX_7500
TX_7600
TX_7700
ReportPoint
ID String
coordinate IPosition
incomingFlight List<ReportPointFields>
FlightPlanFields
reportPoint ReportPoint
localDateTime LocalDateTime
altitude Integer
ReportPointFields
flight AFlight
time LocalDateTime
altitude Integer
Call
flightState Transponder
telNumber String
IgnoreStateStrategy
Powered by yFiles