Airport Simulation

Design and implement a simple airport simulator in C++.

Airport Model

The model is based on following assumptions:

1. An airport can be configured with a number of runways, parking stands, and the operation duration.
2. Each runway is described by a unique identifier and has a single parameter, its length.
3. Each runway can handle one operation at a time. It has a well-defined state: `InOperation`, `Reserved` or `Available`.
4. An operation can be one of two types: landing or take-off. Only one aircraft can be serviced at a time.
5. A parking stand has a unique identifier and can hold only one aircraft at a time. It can be `Occupied`, `Reserved` or `Available`
6. For simplicity, we assume zero time is needed to transfer from the runway to the parking stand (i.e. after the landing procedure is completed, the aircraft immediately appears at the parking stand and vacates the parking stand immediately when the take-off procedure starts).
7. The airport object can receive any number of landing and take-off requests from the aircrafts (number of aircrafts trying to get serviced is not limited in any way).

Implementation Details

`Airport` class needs to provide the following thread safe methods:

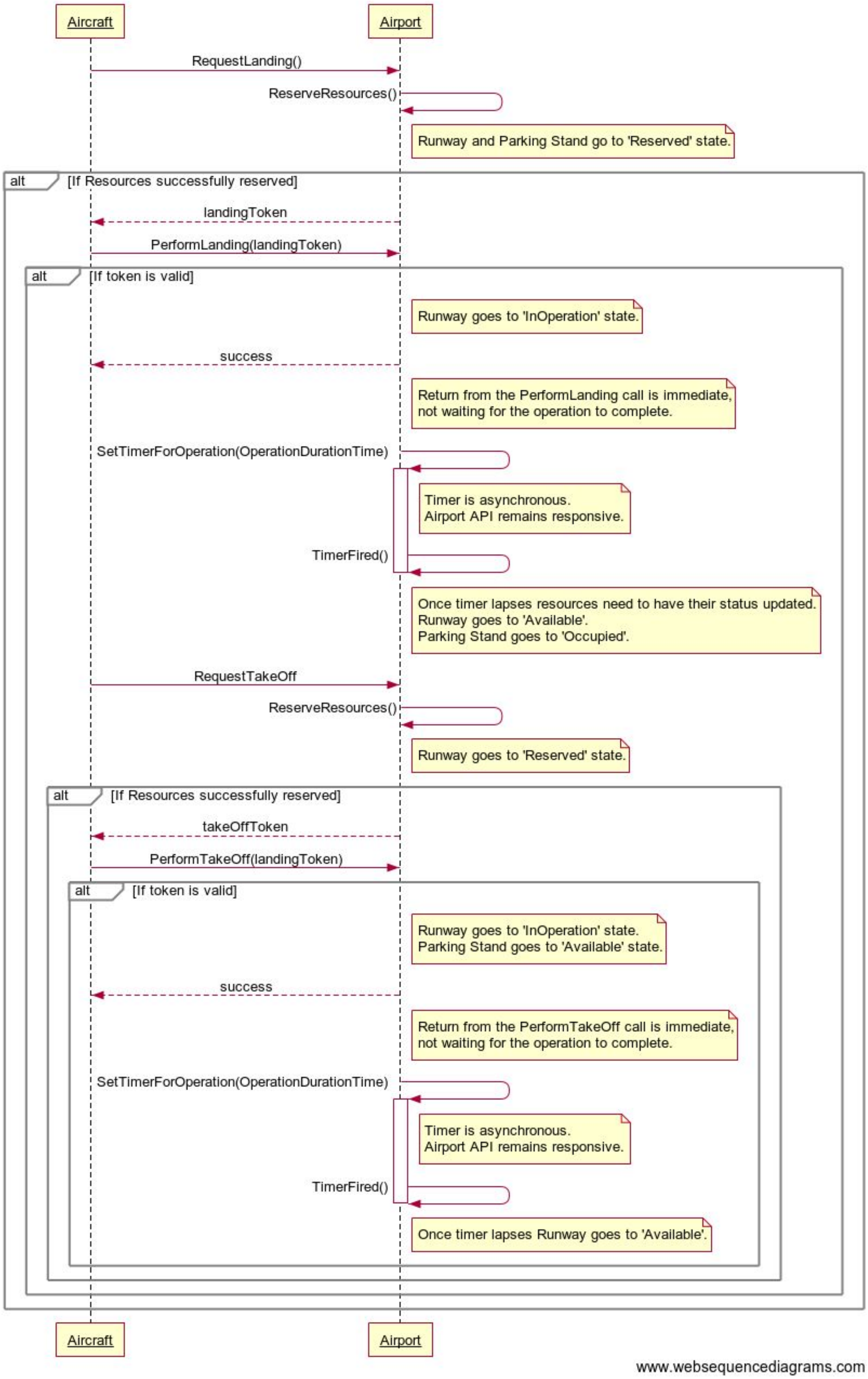
1. RequestLanding	
Comment	After a successful call, the resources necessary to perform the operation should be reserved for the time of the returned authorization.
Input	`aircraftId` - Unique ID of the aircraft. For the sake of simplicity we can assume the uniqueness is provided by the callers.
Output	Returns one of two replies: 1. `Hold` – meaning the airport can't handle the operation at this time. Caller should apply some delay and try again. 2. `Proceed` – landing authorized. In order to issue this reply, the airport needs to have a runway and a parking stand available. Call should also return a `LandingRequestToken` composed of: a. `aircraftId`

	b. `runwayId` c. `parkingStandId` d. `expiration` – a time after which the reservation expires.
2. PerformLanding	
Comment	After this call is executed successfully, the used runway is `InOperation` for the length of operation duration parameter and can't handle any other operations during this time. After the operation is completed, the runway becomes `Available` again, and the reserved parking stand becomes `Occupied`. The call is non-blocking, i.e. it returns before the runway is cleared.
Input	A token obtained through a `RequestLanding` call.
Output	A success flag or error information in case of a failure (e.g. invalid parameters, expired token...).
3. RequestTakeOff	
Comment	After successfull call, the resources necessary to perform the operation should be reserved for the time of the returned authorization.
Input	`aircraftId` - Unique ID of the aircraft. For the sake of simplicity we can assume the uniqueness is provided by the callers.
Output	Returns one of two replies: 1. `Hold` – meaning the airport can't handle the operation at this time. Caller should apply some delay and try again. 2. `Proceed` – take-off authorized. In order to issue this reply, the airport needs to have a runway available. Call should also return a `TakeOffRequestToken` composed of: a. `aircraftId` b. `runwayId` c. `expiration` – a time after which the reservation expires.
4. PerformTakeOff	
Comment	After this call is executed successfully, the occupied parking stand becomes `Available,`and the used runway is `InOperation` for the length of operation duration parameter and can't handle any other operations during this time. After operation finishes, the runway becomes `Available` again. The call is non-blocking, i.e. it returns before the runway is cleared.
Input	A token obtained through a `RequestTakeOff` call.
Output	A success flag or error information in case of a failure (e.g. invalid parameters, expired token...).

Sequence diagram

For simplicity the diagram below depicts an ideal flow, where all calls are successful. Error conditions have to be considered as well. The diagram suggests a solution based on timers to achieve non-blocking of the Airport API and immediate return from Perform[...] methods. Alternative approaches are valid as long as they meet the requirements above.

Airport Simulation Operation



Runway and Parking Stand go to 'Reserved' state.

alt
[If Resources successfully reserved]
landingToken
PerformLanding(landingToken)
alt
[If token is valid]

Runway goes to 'InOperation' state.

Return from the PerformLanding call is immediate,
not waiting for the operation to complete.

Timer is asynchronous.
Airport API remains responsive.

Once timer lapses resources need to have their status updated.
Runway goes to 'Available'.
Parking Stand goes to 'Occupied'.

Runway goes to 'Reserved' state.

alt
[If Resources successfully reserved]
takeOffToken
PerformTakeOff(landingToken)
alt
[If token is valid]

Runway goes to 'InOperation' state.
Parking Stand goes to 'Available' state.

Return from the PerformTakeOff call is immediate,
not waiting for the operation to complete.

Timer is asynchronous.
Airport API remains responsive.

Once timer lapses Runway goes to 'Available'.

www.websequencediagrams.com

Bonus Points

Tackle the following problems if time and energy allow:

1. Provide reasonable code coverage with unit tests. Stress test the implementation with emphasis on the thread safety. Test edge cases.
2. Assert uniqueness of aircraft identifiers (which data structures would you use?).
3. Find solution for methods returning complex data structures (success flag, error info and actual results).
4. Ensure that reserved resources are released back to the pool if the request token is never used.