1 The project

The project is to construct a specification, refinement and implementation of an airline reservation system.

1.1 Brief Overview

The airline reservation system maintains information on flights, classes of seats, personal preferences, prices and bookings. The system allows three classes of users: casual, registered and administrators; to view flights on particular days and to book a number of seats. Before finalising the booking of a flight, personal information must have been entered. Registered users may go on a waiting list for seats if the desired flight does not have enough free seats. Management of the waiting list is left undefined. Registered users must logon using a userid and password, and the system maintains personal information for such users. Registered users may view and update their personal information.

Optional capabilities: Check-in, Overbooking.

2 Requirements

2.1 Concepts to be Modelled

The system must model the following:

flights: a flight is characterised by a point of departure and point of arrival, classes of seating and the number of seats within each class.

seats: available seats in each class on particular flights.

seatprices: prices of seats according to class.

bookings: seats booked by users.
preferences: user preferences.

userids and passwords: for registered users.

personal information: personal information for registered users.

waiting lists: if seats are not available on a particular flight, a registered user may go onto a waiting queue.

dates and times: used for flights.

2.2 Operations

\textit{status} \leftarrow \textit{CasualLogon}: logon as a casual user. No id or password required.

\textit{status} \leftarrow \textit{RegisteredLogon(userid, password)}: Logon as a registered user.

\textit{status, flightno, departure, arrival, price, searchid} \leftarrow \textit{ViewFreeFlight(class, date, timeslot, from, to, numberofseats)}: search for flights with available seats, and return details of one flight and search identifier.

\textit{status, flightno, departure, arrival, price, searchid} \leftarrow \textit{ViewNextFreeFlight}: return details of another flight with available seats and search identifier.

\textit{status, flightno, departure, arrival, freeseats, searchid} \leftarrow \textit{ViewFlight(class, date, timeslot, from, to)}: search for flights regardless of available seats, and return details of one flight and search identifier.

\textit{status, flightno, departure, arrival, freeseats, searchid} \leftarrow \textit{ViewNextFlight}: return details of another flight, regardless of available seats and search identifier.

\textit{status, cost, transid} \leftarrow \textit{BookSeats(searchid)}: book seats on flight corresponding to searchid. Supply a transaction identifier.

\textit{status} \leftarrow \textit{PersonalInfo(transid, name, address, phone, email)}: Supply name and contact information, for non-registered user.

\textit{status} \leftarrow \textit{PassengerName(transid, ordinal, name)}: Supply name of passenger number \textit{ordinal}.

\textbf{Note:} a booking of \textit{n} seats corresponds to \textit{n} passengers that can be given ordinals in 1..\textit{n}.

\textit{status} \leftarrow \textit{PassengerPref(transid, ordinal, preference)}: specify preference for passenger number \textit{ordinal}.

\textit{status, bookingid} \leftarrow \textit{PaySeats(transid, cardnumber, cardname, expirydate)}: pay for booked seats.

\textit{status, waitingid} \leftarrow \textit{AddToWaiting(searchid, numberofseats)}: go on the waiting list.

\textit{status, searchid} \leftarrow \textit{CheckWaiting(waitingid)}: check waiting. The status will indicate whether the seats are now available. If they are then the user can proceed with BookSeats.

\textit{status} \leftarrow \textit{BeginRegister(name, password)}: begin registration as a registered user. This gives the user a temporary logon. User must update address, phone numbers and email and then run EndRegister.

\textit{status} \leftarrow \textit{UpdateAddrress(address)}: update address of registered user. Must be logged on as a registered user, via the operations Logon or Register.
status ← UpdateHomePhone(home): update home phone number of registered user.
status ← UpdateWorkPhone(work): update work phone number of registered user.
status ← UpdateMobilePhone(mobile): update mobile phone number of registered user.
status, userid ← EndRegister(name, password): finish registration as a registered user.
status, flightno ← NewFlight(departure, arrival): create a new flight between departure and destination.
status ← ScheduleFlight(flightno, departure, arrival): add a flight, specified by flight number, on a particular date and departure and arrival times. Perhaps you might want an operation (or operations) that repeats a scheduled flight between two dates on daily or weekly intervals?
status ← ConfigureFlight(flightno, class, seats): configure the class seating number for a flight.
status ← CancelFlight(date, flightno): cancel a flight on a particular date. What do we do with bookings?
status ← SetPrice(flightno, class, price): set the price of seats in a particular class on a particular flight number. Note: in real airlines it is much more complicated than this!

2.3 Modelling Sets, Constants and Definitions
Context machines should be used for sets, constants and definitions.
You will need to model date.

3 What you should do
1. Develop “low-level” machines with preconditioned, fragile operations.
2. Animate the specifications and develop animation scripts that illustrate the checking of the various conditions in the requirements.
3. Discharge the proof obligations.
4. Develop robust specifications.
5. Discharge the proof obligations.
6. Develop refinements.
7. Discharge proof obligations.
8. Develop implementations.
9. Discharge as many proof obligations as possible.

4 The Challenge
The challenge of this assignment is to resist any temptation to rush to implementation, and to work your way through the refinements, implementations and associated proof obligations. This way you will experience the novelty of generating code that must be correct, modulo the correctness of the specification.
Good luck!