Airline Reservation System

Modules

1. Airline_Reservation_System
2. Travel_Agent
3. Airline_manager
4. Date (Directly Implemented)

I. Airline_Reservation_System
1. Top Level of the Architectural Design

MACHINE airline_reservation_system_design
   PRAGMA task(airline_reservation_system_design)
   INHERIT airline_reservation_system
   INHERIT time
   INHERIT time_unit

   TEMPORAL retire --time even occurs a 4 a.m.
   WHERE hour(time) = 4 & PERIOD = (1 days)
   TRANSITION ALL (r: reservation ::
       r: IN reservations <=>
       r: IN reservations & date(r) >= date(time) )

END

2. Specification for Flight_Schedule

TYPE flight_schedule
   INHERIT final_flight_view
   -- Flight concepts from the environment model.
   IMPORT flight_description
       FROM final_travel_agent_interface

   MODEL(m: map(flight_id, flight), name: string)
   INVARIANT ALL(fs: flight_schedule :: same_id(fs)).
   ALL(fs1 fs2: flight_schedule :: fs1.name = fs1.name => fs1 = fs2 )
   -- PRAGMA persistent (flight_schedule)
   -- PRAGMA direct_allocation(flight_schedule)

   MESSAGE open(name: string, fs: flight_schedule)
       -- PRAGMA update(fs, fs1)
   WHEN SOME(fs1: flight_schedule :: fs1.name = name)
       REPLY (fs1: flight_schedule) WHERE fs1.name = name
   OTHERWISE REPLY (fs1: flight_schedule)
       TRANSITION new(fs1), domain(fs1.m) = fs1.name = name
MESSAGE add(i: flight_id,  
    price: money,  
    origin_destination: airport,  
    departure_arrival: time,  
    capacity: nat  
    fs: flight_schedule)  
    CHOOSE(f:flight SUCH THAT id(f) = i & price(f) = price &  
        origin(f) = origin & destination(f) = destination &  
        departure(f) = departure & arrival(f) = arrival &  
        capacity(f) = capacity )  
    TRANSITION fs.m = bind(i, f, *fs.m) -- Add flight  
    -- replaces the previous flight if there was one.

MESSAGE remove (i: flight_id, fs: flight_schedule)  
    TRANSITION fs.m = remove(i, fs.m)

MESSAGE member(i: flight_id, fs: flight_schedule)  
    REPLY (b: boolean) WHERE b <=> i IN fs.m

MESSAGE capacity(i: flight_id, fs: flight_schedule)  
    WHEN i IN domain(fs.m)  
        REPLY (n:nat) WHERE n = capacity(fs.m[i])  
    OTHERWISE REPLY EXCEPTION no_such_flight

MESSAGE find_flights(fs: flight_schedule, o d: airport)  
    GENERATE(s: sequence(flight_description)  
        WHERE ALL(f: flight :: description(f) IN s <=>
            f IN range(fs.m) & origin(f) = o & destination(f) = d)

CONCEPT same_id(fs: flight_schedule)  
    VALUE (b: boolean)  
    WHERE b <=> ALL(i:  flight_id SUCH THAT i IN fs.m ::
        id(fs.m[i]) = i )

END airline_reservation_system_design

### 2a. Intended Use of Flight Schedule Operations

<table>
<thead>
<tr>
<th>Arline Reservation System</th>
<th>Flight_Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIALLY</td>
<td>Open</td>
</tr>
<tr>
<td>Find_Flights</td>
<td>Find_Flights</td>
</tr>
<tr>
<td>Reserve</td>
<td>Capacity</td>
</tr>
<tr>
<td>Cancel</td>
<td>Member</td>
</tr>
<tr>
<td>Notify</td>
<td>Member</td>
</tr>
<tr>
<td>Add_flight</td>
<td>Member add</td>
</tr>
<tr>
<td>Drop_flight</td>
<td>Member, remove</td>
</tr>
<tr>
<td>New_fare</td>
<td>Member, add</td>
</tr>
<tr>
<td>Retire</td>
<td></td>
</tr>
</tbody>
</table>
3. Specification for Reservation_Set

TYPE reservation_set
    INHERIT final_travel_agent_view
    -- Reservation properties from environment model
    IMPORT single_reservation FROM final_travel_agent_interface
    IMPORT holds_concept bookings_concept
        FROM airline_reservation_system_design1

MODEL (s: set(reservation), name:string)
INVARIANT ALL (rs: reservation_set :: single_reservation(rs.s))
--PRAGMA persistent(reservation_set)
--PRAGMA direct_allocation(reservation_set)

MESSAGE open(name: string, rs: reservation_set)
--PRAGMA update(rs, rs1)
--PRAGMA representation(string, text)
WHEN SOME (rs1: reservation_set :: rs1.name = name)
    REPLY (rs1: reservation_set)
OTHERWISE REPLY (rs1: reservation_set)
    TRANSITION new(rs1), rs1.s = ( ), rs1.name = name

MESSAGE add(i : flight_id, d: date, p:passenger, a:agent, rs: reservation_set)
    CHOOSE(r: reservation SUCH THAT
        id(r) = i & date(r) = d & passenger(r) = p &
        agent(r) = a )
    TRANSITION rs.s = *rs.s U (r)

MESSAGE remove(i : flight_id, d: date, p:passenger, rs: reservation_set)
    CHOOSE(r: reservation SUCH THAT
        r IN *rs.s & passenger(r) = p & id(r) = i &
        date(r) = d )
    TRANSITION rs.s = *rs.s - (r)

MESSAGE remove(i : flight_id, rs: reservation_set)
    TRANSITION ALL (r: reservation :: r IN rs <=>= r IN *rs & id(r) ~= i)

MESSAGE remove(i : flight_id, rs: reservation_set)
    TRANSITION ALL (r: reservation :: r IN rs <=>= r IN *rs & date(r) >= d)

MESSAGE holds(p: passenger, i: flight_id, d: date, rs: reservation_set)
    REPLY(b: boolean)
    WHERE b <=>= holds_concept(p, i, d, rs.s)

MESSAGE bookings(i: flight_id, d: date, rs: reservation_set)
    REPLY(n: nat)
WHERE n = bookings_concept(i, d, rs)

MESSAGE notify(a: agent_id, i: flight_id, d: date, rs: reservation_set)
GENERATE (s: sequence(reservation))
WHERE ALL (r: reservation :: r IN s <=>
    r IN rs & agent(r) = a & id(r) = i &
    date(r) >= d )

END reservation_set

3a. Intended Use for Reservation_Set

<table>
<thead>
<tr>
<th>Arline Reservation System</th>
<th>Reservation_Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIALLY</td>
<td>Open</td>
</tr>
<tr>
<td>Find Flights</td>
<td>Add, holds, bookings</td>
</tr>
<tr>
<td>Reserve</td>
<td>Remove #1, holds</td>
</tr>
<tr>
<td>Cancel</td>
<td>Notify</td>
</tr>
<tr>
<td>Notify</td>
<td></td>
</tr>
<tr>
<td>Add_flight</td>
<td>Remove #2</td>
</tr>
<tr>
<td>Drop_flight</td>
<td></td>
</tr>
<tr>
<td>New_fare</td>
<td>Remove #3</td>
</tr>
<tr>
<td>Retire</td>
<td></td>
</tr>
</tbody>
</table>

4. Renamed Concepts from the Functional Specification (omitted)

5. Specification of B-Tree (omitted)

6. Specification of Lexicographic Ordering (omitted)

7. Refinement of the Flight Schedule
The order in which the flights are displayed by the find_flights command was not
specified previously. Therefore, we will generate the flights with flight_ids in
alphabetical order, since this ordering is provided naturally by the data structure.

TYPE flight_schedule1
INHERIT flight_schedule

MESSAGE find_flights(fs: flight_schedule, o, d: airport)
GENERATE (s: sequence (flight_description) )
WHERE ALL (i, j: nat SUCH THAT
    i IN domain(s) & j IN domain(s) ::
    s[i].id <= s[j].id)

END flight_schedule1

8. Specification for Hashed_File (omitted)
9. Specification of String Hash (omitted)

10. Extension of the Type Date

TYPE date_design
    INHERIT date
    IMPORT nat FROM format
    --PRAGMA unique_representation(date)
    MESSAGE string_code(d1: date) REPLY(s:string)
    WHERE SOME (m d y : string ::
        s = m || d || y & nat(m) = d1.month &
        nat(d) = d1.day & nat(y) = d1.year &
        length(m) = length(d) = length(y) = 2 )
END date_design

11. Algorithm Sketch for Remove #2

procedure remove(i: flight_id, rs: reservation_set)
    first_key := MINIMUM(k: key SUCH THAT k.id = i :: k
    last_key := MAXIMUM(k: key SUCH THAT k.id = i :: k
    for d: date in scan(date_set) loop
        for r: reservation in range (first_key, last_key, rs) loop
            remove (r.p, r.i, d, rs)
        end loop
    end loop
end remove

12. Algorithm Sketch for notify

procedure notify(a:agent_id, i:flight_id, d:date, rs:reservation_set)
    first_key := MINIMUM(k: key SUCH THAT k.id = i :: k
    last_key := MAXIMUM(k: key SUCH THAT k.id = i :: k
    for d1: date in scan_from(d, date_set) loop
        for r: reservation in range (first_key, last_key, rs) such that r.a = a
            loop
            generate(r)
            end loop
    end loop
end notify
II. Travel_Agent

This module must be partitioned for concurrency. The processing performed by each instance of the travel agent is minimal and should not cause performance problems. The travel agent is a generic module with multiple distinct instances. The travel agents will be using telephone connections, and the system must be able to support 300 travel agents each on a separate telephone connection.

13. Implementation of Type Concepts

MACHINE airline_reservation_system_design2
   INHERIT airline_reservation_system_design1
   --PRAGMA representation(sequence(flight_description),
      bounded_sequence (flight_description, 100) )
   --PRAGMA representation(sequence(reservation),
      bounded_sequence (reservation, 1000) )
   --PRAGMA representation(flight_description, record)
   --PRAGMA representation(reservation, record)
END airline_reservation_system_design2

14. Specification of Unbounded Sequences (omitted)

15. Specification of Bounded Sequences (omitted)

16. Top-Level of the travel Agent Module

MACHINE travel_agent_design(modem_id: nat)
   PRAGMA task (travel_agent_design)
   INHERIT final_travel_agent(modem_id)

   STATE(input output : text_file)
   INVARIANT  name(input) = input_name(modem_id)
              name(output) = output_name(modem_id)
              mode(input) = #in_file, mode(output) = #out_file
   INITIALLY is_open(input), is_open(output)

   PRAGMA representation (string, text)
   PRAGMA representation (text_file, text_io.file_type)
   PRAGMA external_input (interpret_command)
   PRAGMA returned_value (flights)
   PRAGMA returned_value (reservations)
   PRAGMA returned_value (done)

   CONCEPT text_file : type
CONCEPT input_name(id : nat) VALUE (s: string)
   -- The file name for the input stream is generated from the model_id

CONCEPT output_name(id : nat) VALUE (s: string)
   -- The file name for the output stream is generated from the model_id

END travel_agent_design

17. Specification for Edit_Command

FUNCTION edit_command
   INHERIT basic_edit -- defines the concept edit.
   MESSAGE (s1 : string) REPLY (s2 : string) WHERE s2 = edit (s1)
   -- PRAGMA representation (string, text)
END

18. Specification for Command Parsing Functions

FUNCTION parse_find_flights
   INHERIT travel_agent_command_formats
   INHERIT airline_reservation_system_type_formats
   MESSAGE (s:string) – PRAGMA representation(string, text)
   WHEN SOME (o d: airport :: is_find_flights("f" || s, o, d) )
      REPLY(b: boolean, o d : airport)
         WHERE b = true, is_find_flights(s,o,d)
   OTHERWISE REPLY (b:boolean, o, d : airport)
      WHERE b = false
END parse_find_flights

FUNCTION parse_reserve
   INHERIT travel_agent_command_formats
   INHERIT airline_reservation_system_type_formats
   MESSAGE (s:string) – PRAGMA representation(string, text)
   WHEN SOME (i: flight_id, d: date, p: passenger :: is_reserve("r" || s, a, i, d, p) )
      REPLY (b:boolean, a:agent_id, i:flight_id, d:date, p;passenger)
         WHERE b = true, is_reserve(s, a, i, d, p)
   OTHERWISE REPLY (b:boolean, a:agent_id, i:flight_id, d:date, p;passenger)
      WHERE b = false
END parse_reserve

FUNCTION parse_cancel
   INHERIT travel_agent_command_formats
   INHERIT airline_reservation_system_type_formats
   MESSAGE (s:string) – PRAGMA representation(string, text)
WHEN SOME (i: flight_id, d: date, p: passenger :: is_cancel("c" || s, i, d, p) )
    REPLY (b:boolean, i:flight_id, d:date, p:passenger)
        WHERE b = true, is_cancel(s, i, d, p)
    OTHERWISE REPLY (b:boolean, i:flight_id, d:date, p:passenger)
        WHERE b = false
END parse_cancel

FUNCTION parse_notify
    INHERIT travel_agent_command_formats
    INHERIT airline_reservation_system_type_formats
    MESSAGE (s:string) – PRAGMA representation(string, text)
        WHEN SOME (a: agent_id, i: flight_id, d:date, p:passenger ::
            is_notify("n" || s, a, i,d) )
        REPLY (b:boolean, i:flight_id, d:date, a:agent_id)
            WHERE b = true, is_notify(s, a, i, d)
        OTHERWISE REPLY (b:boolean, i:flight_id, d:date, a: agent_id)
            WHERE b = false
END parse_notify

19. Specification for Display Functions

FUNCTION display_flight
    INHERIT travel_agent_output_formats
    IMPORT flight_description FROM final_travel_agent_interface
    IMPORT fill FROM format

    MESSAGE (fd: flight_description, f: text_file)
        -- PRAGMA nonlocal_variable(f)
        -- PRAGMA representation(text_file, text_io.file_type)
        SEND put_line(f: text_file, s: string) TO text_io
            WHERE s = fill(flight_row(fd), [8,8,8,10] )
END display_flight

FUNCTION display_reservation
    INHERIT travel_agent_output_formats
    IMPORT reservation FROM airline_reservation_system_design
    IMPORT fill FROM format

    MESSAGE (r: reservation, f: text_file)
        -- PRAGMA nonlocal_variable(f)
        -- PRAGMA representation(text_file, text_io.file_type)
        SEND put_line(f: text_file, s: string) TO text_io
            WHERE s = fill(reservation_row(r), [8,20] )
END display_reservation
20. Specification for Next_Item

FUNCTION next_item
IMPORT no_spaces FROM character_properties

MESSAGE (s:string) -- PRAGMA representation(string, text)
REPLY (next rest : string) -- PRAGMA update(s, rest)
   WHERE SOME(b1 b2: string SUCH THAT spaces(b1) & spaces(b2) ::
      s = b1 || next || b2 || rest & no_spaces(next) &
      (b2 = "" => rest = ")
     )

next = "" => spaces(s)
-- Extract the next nonblank string. Next is nonempty unless s has nothing
-- but spaces. The delimiter b2 contains at least one space unless “next”
-- goes to the end of s.

END next_item

21. Specification of Type Recognition Functions

FUNCTION is_airport -- PRAGMA representation(string, text)
   INHERIT airline_reservation_type_formats
   MESSAGE (s:string) REPLY (b: boolean) where b <=> s IN airport
END is_airport

FUNCTION is_flight_id -- PRAGMA representation(string, text)
   INHERIT airline_reservation_type_formats
   MESSAGE (s:string) REPLY (b: boolean) where b <=> s IN flight_id
END is_flight_id

FUNCTION is_passenger -- PRAGMA representation(string, text)
   INHERIT airline_reservation_type_formats
   MESSAGE (s:string) REPLY (b: boolean) where b <=> s IN passenger
END is_passenger

FUNCTION is_agent_id -- PRAGMA representation(string, text)
   INHERIT airline_reservation_type_formats
   MESSAGE (s:string) REPLY (b: boolean) where b <=> s IN agent_id
END is_agent_id

FUNCTION is_date-- PRAGMA representation(string, text)
   INHERIT airline_reservation_type_formats
   MESSAGE (s:string)
      WHEN SOME (d:date :: date = date(s) )
         REPLY(b: boolean, d: date) WHERE b <=> true, d = date(s)
      OTHERWISE REPLY (b: boolean, d:date) where b <=> false
END is_date
22. Specification for Fill_Line

FUNCTION fill_line
    IMPORT fill_row FROM format
    MESSAGE (r: row, w: sequence(nat))
    END fill_row

III. Airline Manager

23. Top-Level Design for the Airline Manager

MACHINE airline_manager_design
    -- PRAGMA task(airline_manager_design)
    INHERIT final_airline_manager
    IMPORT text_file input_name output_name
    FROM travel_agent_design
    STATE(input output : text_file)
    INVARIANT name(input) = input_name(0),
    name(output) = output_name(0),
    mode(input) = #in_file, mode(output) = #out_file
    INITIALLY is_open(input), is_open(output)

    --PRAGMA representation(string, text)
    --PRAGMA representation(text_file, text_io.file_type)
    --PRAGMA external_input(interpret_command)
    --PRAGMA returned_value(flights)
    --PRAGMA returned_value(reservations)
    --PRAGMA returned_value(done)
END airline_manager_design

24. Specifications of Command Parsing Functions

FUNCTION parse_add_flight
    INHERIT airline_manager_command_formats
    INHERIT airline_reservation_system_type_formats
    MESSAGE (s:string) -- PRAGMA representation (string, text)
    WHEN SOME (i: flight_id, p: money, o d : airport, dep arr : time, c: nat ::
    is_add_flight("add_flight " || s, i, p, o, d, dep, arr, c))
    END add_flight
    END parse_add_flight
FUNCTION parse_drop_flight
    INHERIT airline_manager_command_formats
    INHERIT airline_reservation_system_type_formats
    MESSAGE (s:string) -- PRAGMA representation (string, text)
        WHEN SOME (i: flight_id::
            is_drop_flight("drop_flight " || s, i ) )
            REPLY( b: boolean, i: flight_id)
                WHERE b = true is_drop_flights(s, i)
            OTHERWISE REPLY( b: boolean, i: flight_id)
                WHERE b = false
    END parse_drop_flight

FUNCTION parse_new_fare
    INHERIT airline_manager_command_formats
    INHERIT airline_reservation_system_type_formats
    MESSAGE (s:string) -- PRAGMA representation (string, text)
        WHEN SOME (i: flight_id::
            is_new_fare("new_fare " || s, i, p) )
            REPLY( b: boolean, i: flight_id, p:money)
                WHERE b = true is_new_fares(s, i, p)
            OTHERWISE REPLY( b: boolean, i: flight_id, p:money)
                WHERE b = false
    END parse_new_fare

25. Specifications for Type Recognition Functions

FUNCTION is_money -- PRAGMA representation(string, text)
    INHERIT airline_reservation_type_formats
    MESSAGE (s:string) REPLY (b:boolean) WHERE b <=> s IN money
END is_money

FUNCTION is_time -- PRAGMA representation(string, text)
    INHERIT airline_reservation_type_formats
    MESSAGE (s:string) REPLY (b:boolean) WHERE b <=> s IN time
END is_time

IV. Design Review

26. Module Dependency Diagram (to be done by each group)

27. Module Dependency Diagram for the Travel Agent (to be done by each group)

28. Module Dependency Diagram for the Airline Manager (to be done by each group)
V. Concrete Interface Design

29. Concrete Interface for the Airline Reservation System

with bounded_sequence_pkg;
with date_pkg; use date_pkg;
with concrete_types_pkg; use concrete_types_pkg;

package airline_reservation_system_pkg is
  -- Generic instantiations for interface types
  package flight_sequence_pkg is new bounded_sequence_pkg(flight_description);
  package reservation_sequence_pkg is new bounded_sequence_pkg (flight_description)
    use flight_sequence_pkg; use reservation_sequence_pkg;
  subtype flight_sequence is flight_sequence_pkg.sequence;
  subtype reservation_sequence is reservation_sequence_pkg.sequence;

  -- Exceptions.
  flight_exists: exception;
  reservation_exists: exception;
  no_such_flight: exception;
  no_reservation: exception;
  no_seat: exception;

  task airline_reservation_system is
    -- Travel agent interface
    entry find_flights(origin, destination: in airport; fs: out flight_sequence);
    entry reserve(a: in agent_id; i: in flight_id; d: in date; p: in passenger);
    entry cancel(i: in flight_id; d: in date; p: in passenger);
    entry notify(a: in agent_id; i: in flight_id; d: in date;
      rs: out reservation_sequence);

    -- Airline manager interface
    entry add_flight(i: in flight_id; price: in money; origin, destination: in airport;
      departure, arrival: in flight_time; capacity: in natural );
    entry drop_flight(i: in flight_id);
    entry new_fare(i: in flight_id; in price: in money);

    -- Temporals
    entry retire;
  end airline_reservation_system;
end airline_reservation_system_pkg;
30. Definitions of Concrete Ada Types

    with date_pkg; use date_pkg; -- Defines date.
    with text_pkg; use text_pkg;  -- Defines text_pkg_instance.
    package concrete_types_pkg is  -- Concrete type definitions.
      use text_pkg_instance;  -- Defines the type text.
    subtype agent_id is text(3); -- Fixed length.
    subtype airport is text(3); -- Fixed length.
    subtype flight_id is text;  -- Variable length.
    subtype money is text;  -- Variable length.
    subtype flight_time is text;  -- Variable length.
    subtype passenger is text;  -- Variable length.

type flight_description is
  record
    id: flight_id;
    dep, arr: flight_time;
    price: money
  end record;

type reservation is
  record
    id: flight_id;
    d: date;
    p: passenger;
    a: agent_id
  end record;
end concrete_types_pkg;

31. Definition of Text_Pkg

    with generic_text_pkg;
    package text_pkg is
      package text_pkg_instance is
        new generic_text_pkg(max_text_length => 80);
    end text_pkg;

32. Concrete Interface for the Travel Agent

    generic
      modem_id : natural;
    package travel_agent_pkg is
      task travel_agent;
    end travel_agent_pkg;
33. Concrete Interface for the Airline Manager

```vhdl
package airline_manager_pkg is
    task airline_manager;
end airline_manager_pkg;
```

34. Interfaces for Travel Agent Auxiliary Operations

```vhdl
with text_pkg; use text_pkg; -- Defines text_pkg_instance.
with date_pkg; use date_pkg; -- Defines date.
with concrete_types_pkg; use concrete_types_pkg;
    --defines airport, flight_id, passenger, agent_id, flight_description, reservation.
with parse_pkg, use parse_pkg;
    --defines is_airport, is_flight_id, is_passenger, is_agent_id, edit_command, next_item.

package body travel_agent_pkg is
    use text_pkg_instance; defines the type text.

    --Local subprogram declarations.
    procedure parse_find_flights(s: in text; o,d airport; result: out boolean);
    procedure parse_reserve(s: in text; a: out agent_id; i: out flight_id; d: out date;
                           p: out passenger; result: out boolean);
    procedure parse_cancel(s: in text; i: out flight_id; d: out date;
                           p: out passenger; result: out boolean);
    procedure parse_notify(s: in text; a: out agent_id; i: out flight_id; d: out date;
                           result: out boolean);
end travel_agent_pkg;
```

35. Interfaces for Airline Manager Auxiliary Operations

```vhdl
with text_pkg; use text_pkg; -- Defines text_pkg_instance.
with date_pkg; use date_pkg; -- Defines date.
with concrete_types_pkg; use concrete_types_pkg; -- Defines airport, flight_id, money,
    -- flight_time.
    with parse_pkg; use parse_pkg;
    -- Defines is_airport, is_flight_id, is_money, is_time, edit_command, next_item.

package body airline_manager_pkg is
    use text_pkg_instance; -- Defines the text type.

    -- Local subprogram declarations.
    procedure interpret_command(cmd: string; args: in text; f: in file_type);
    procedure parse_add_flights(s: in text; i: out flight_id; p: out money; o, d: out airport;
                               dep, arr: out flight_time; c: out natural; result: out boolean);
    procedure parse_drop_flights(s: in text; i: out flight_id; result: out boolean);
    procedure parse_new_fare(s: in text; i: out flight_id; p: out money; result: out boolean);
end airline_manager_pkg
```
36. Interfaces for Shared Auxiliary Operations

with text_pkg; use text_pkg; -- Defines text_pkg_instance.
with date_pkg; use date_pkg; -- Defines date.
package parse_pkg is
  use text_pkg_instance; -- Defines the text type.
    function edit_command(s: text) return text;
    procedure next_item(s: in out text; next: out text);
    function is_airport(s: text) return boolean;
    function is_flight_id(s: text) return boolean;
    function is_passenger(s: text) return boolean;
    function is_agent_id(s: text) return boolean;
    procedure next_item(s: in out text; next: out text);
    function is_money(s: text) return boolean;
    function is_time(s: text) return boolean;
end parse_pkg

37. Concrete Interface for Date

with text_pkg; use text_pkg; -- Defines text_pkg_instance.
package date_pkg is
  use text_pkg_instance; defines the type text.
    type date_type is private;
    function create(d, m, y: natural) return date;
    function “<”(di, d2: date) return boolean;
    function “>”(di, d2: date) return boolean;
    function string_code(d: date) return text;
    illegal_date: exception; -- raised by create.
private
  type date is ?;
end date_pkg;

38. Concrete interface for Flight Schedule

with text_pkg; use text_pkg; -- Defines text_pkg_instance.
with concrete_types_pkg; use concrete_types_pkg; -- Defines airport, flight_id, money, flight_time, flight_description.
package flight_schedule_pkg is limited private;

  procedure open(name: in text; fs: in out flight_schedule);
  procedure add(i: in flight_id; price: in money; origin, destination: in airport;
departure, arrival: in flight_time; capacity: in natural;
fs: in out flight_schedule);
procedure remove(i: in flight_id; fs: in out flight_schedule);
procedure member(i: in flight_id; fs: in out flight_schedule; result: out boolean);
procedure capacity(i: in flight_id; fs: in out flight_schedule; result: out natural);
generic
with procedure produce(fd: in flight_description);
procedure find_flights(fs: in out flight_schedule; origin, destination: airport);
no_such_flight: exception -- Raised by capacity.

private
  type flight_schedule is ?;
end flight_schedule_pkg;

39. Concrete Interface for Reservation Set
with concrete_types_pkg; use concrete_types_pkg;
with text_pkg; use text_pkg; -- Defines text_pkg_instance.
with date_pkg; use date_pkg; -- Defines date.

package reservation_set is
  use text_pkg_instance; defines the type text.

  type reservation_set is limited private;

  procedure open(name: in text; rs: in out reservation_set);
  procedure add(a: in agent_id; i: in flight_id; d: in date; p: in passenger;
                   rs: in out reservation_set);
  procedure remove (i: in flight_id; d: in date; p: in passenger; rs: in out reservation_set);
  procedure remove(i: in flight_id; rs: in out reservation_set);
  procedure remove(d: in date; rs: in out reservation_set);
  procedure holds( p: in passenger; i: in flight_id; d: in date; rs: in out reservation_set; has: out boolean);
  procedure bookings(i: in flight_id; d: in date; rs: in out reservation_set;
                      number_of_bookings: out natural);

generic
  with procedure(r: in reservation);
  procedure notify(a: in agent_id; i: in flight_id; d: in date; rs: in out reservation_set);

private
  type reservation set is ?;
end reservation_set;