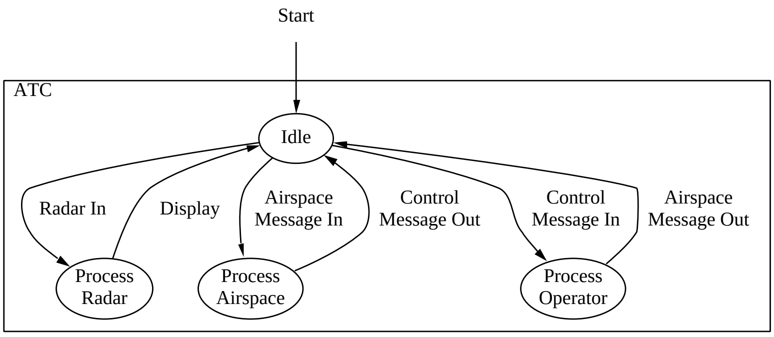
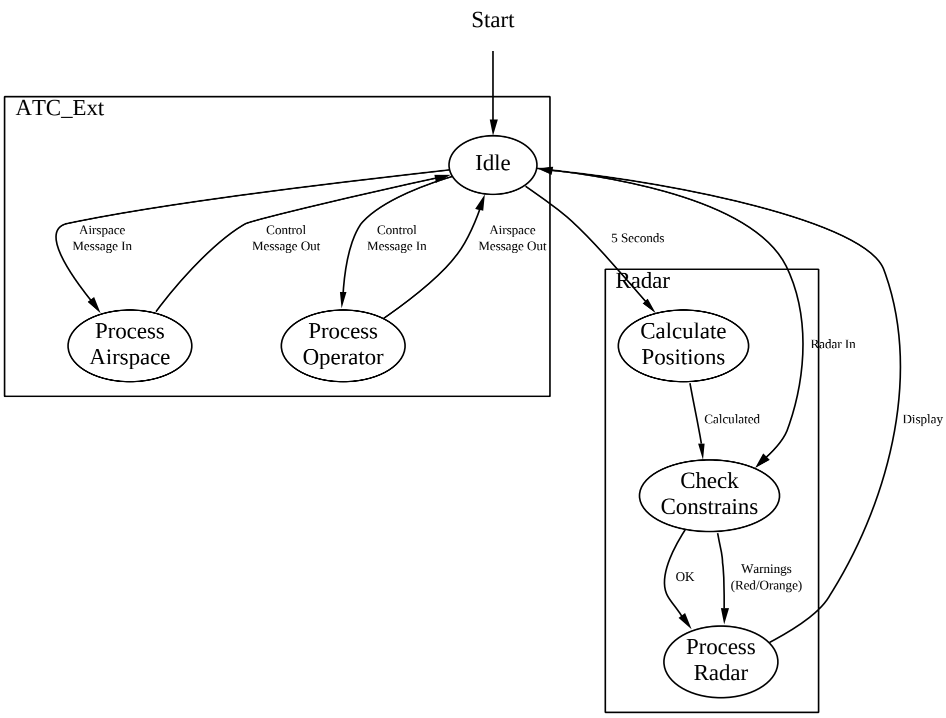
Exercise 2002/2003
Air Traffic Control

Part 1

I desided to leave the concurrency out of the diagramm to increase the clarity. The three message cycles are in parallel. The simple version is illustrated in the following state chart diagram:



In the folloing diagram all the required extentions are built in. The message queues and the global knowlegde are hidden.



To prevent the case that `Radar In` occurs short after or for the 5 seconds timer I decided to do this compution in serial. The dotted arrows are messages without changing the state. (sorry image file corrupted)

Part 2

Stage 1

Simple Broadcast the messages.

AtcControl.strl.Stage1

```
%Defines the module for the outermost control structure for the ATC
%Reactive Computer System.
%

module AtcControl:
type Vector;
type RadarData;
type Aircraft;
type Message;

input radar:RadarData;
input inAirspaceMessage:Message;
input inControlMessage:Message;

output display:RadarData;
output outControlMessage:Message;
output outAirspaceMessage:Message;

every radar do
  emit display(?radar)
end every
||
every inAirspaceMessage do
  emit outControlMessage(?inAirspaceMessage)
end every
||
every inControlMessage do
  emit outAirspaceMessage(?inControlMessage)
end every

end module
```

Stage 2

I added a new module `SyncRadar` to the code and a variable to copy the radardata and to know which airplanes are new respective obsolete. The communication is done by a local signal calles `S_airplanes`.

AtcControl.strl.Stage2

```
% Defines the module for the outermost control structure for the ATC
% Reactive Computer System.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% module for Syncronization of radardata and intern airplanes
%  Assumption: S_airplanes is emited before
module SyncRadar:
type Vector;
type Aircraft;
type RadarData;
type Message;

 %% A declaration of each procedure implemented in AtcControlInterface.java
procedure makeVector (Vector) (integer, integer, integer);
procedure updateAircraftPositionInRadarData (RadarData) (Aircraft);
procedure addAircraftToRadarData (RadarData) (Aircraft);
procedure removeAircraftFromRadarData (RadarData) (Aircraft);
procedure removeRadarDataIteratorItem () ();
```

```
procedure addMessageTargetItem()(Aircraft);
procedure removeMessageTargetItem()(Aircraft);
procedure setAircraftVelocity (Aircraft) (Vector);
procedure setAircraftColour (Aircraft) (Vector);

%% A declaration of each function implemented in AtcControlInterface.java
function initialiseRadarDataIterator (RadarData) : integer;
function nextRadarDataIteratorItem () : Aircraft;
function radarDataContainsAircraft(RadarData, Aircraft) : boolean;
function getAircraftFromRadarData (RadarData, string) : Aircraft;
function isAircraftNull (Aircraft) : boolean;
function getAircraftID (Aircraft) : string;
function getAircraftPosition (Aircraft) : Vector;
function getAircraftVelocity (Aircraft) : Vector;
function getAircraftColour (Aircraft) : Vector;
function hasNextRadarDataIteratorItem () : boolean;

input radar : RadarData;
inputoutput S_airplanes : RadarData;

  % Update newcomers or leaved planes
  var
      airplanes      : RadarData,
      msg           : Message,
      src           : string,
      number         : integer,
      aircraft       : Aircraft
  in
    airplanes := ?S_airplanes;

    % Go through all planes in our my airplanes and look if there are in radardata as well
    number := initialiseRadarDataIterator (airplanes);
    repeat number times
        if hasNextRadarDataIteratorItem() then
          aircraft := nextRadarDataIteratorItem ();
          % is this not in radardata?
          if not radarDataContainsAircraft(?radar, aircraft) then
            % through airplane out of local radar and Message Control
            call removeRadarDataIteratorItem()();
            call removeMessageTargetItem ()(aircraft);
          end if
        end if;
        pause;
    end repeat;

    % Go through all planes in radardata and look if there are in my airplanes var
    number := initialiseRadarDataIterator (?radar);
    repeat number times
        if hasNextRadarDataIteratorItem() then
          aircraft := nextRadarDataIteratorItem ();
          % is this in my airplanes?
          if radarDataContainsAircraft(airplanes, aircraft) then
            % update pos
            call updateAircraftPositionInRadarData (airplanes) (aircraft);
          else
            % add aircraft to local radardata and to Messageinterface
            call addMessageTargetItem ()(aircraft);
            call addAircraftToRadarData (airplanes) (aircraft);
          end if
        end if;
        pause;
    end repeat;

    pause;
    emit S_airplanes(airplanes);
  end var;
end module
```

```


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
module AtcControl :

type Vector;
type Aircraft;
type RadarData;
type Message;

input radar:RadarData;
input inAirspaceMessage:Message;
input inControlMessage:Message;
output display:RadarData;
output outControlMessage:Message;
output outAirspaceMessage:Message;

var airplanes      : RadarData
in
  signal S_airplanes   : RadarData
  in
    emit S_airplanes(airplanes);
    every radar  do
        % Synchronize radar and airplanes
        run SyncRadar;
        % get results
        airplanes := ?S_airplanes;
        emit display(airplanes);
    end every
  end signal
end var
  ||
every inAirspaceMessage do
  emit outControlMessage(?inAirspaceMessage)
end every
  ||
every inControlMessage do
  emit outAirspaceMessage(?inControlMessage)
end every

end module

```

Stage 3

To get the velocities I wrote a module called `MessageHandler` that process the Handoff messages and the Report messages which are in queue A. To deal with the fact that I get a Handoff message without having the airplane allready, I established a new variable of `RadarData` called `commingplanes`. However to add there a new `Aircraft` I needed a function to set an ID of an Aircraft. So I added the function `newAirCraft` to `AtcControlInterface.java` (see end of Stage 4). If now a new plane is comming, then I look in the `commingplanes` if I find it and take the velocity over.

To predict the positions I wrote a module `PredictPositions` that adds for each plane the fivefold of the velocity vector to the position vector.

AtcControl.strl.Stage3

```
% Defines the module for the outermost control structure for the ATC
% Reactive Computer System.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% module for handling Messages in the queues
```

```
%  Assumption: S_airplanes and S_comingplanes are emited before
module MessageHandler:
type Vector;
type Aircraft;
type RadarData;
type Message;
procedure enqueueOnA ()(Message);
procedure enqueueOnB ()(Message);
procedure setAircraftPosition (Aircraft) (Vector);
procedure setAircraftVelocity (Aircraft) (Vector);
procedure setAircraftColour (Aircraft) (Vector);
procedure addAircraftToRadarData (RadarData) (Aircraft);
function dequeueFromA () : Message;
function dequeueFromB () : Message;
function isListAEmpty () : boolean;
function isListBEmpty () : boolean;
function isHandoffMessage (Message) : boolean;
function isReportMessage (Message) : boolean;
function getMessageSrc (Message) : string;
function getMessageTgt (Message) : string;
function getMessageReference (Message) : string;
function getMessageTime (Message) : integer;
function getMessagePosition (Message) : Vector;
function getMessageVelocity (Message) : Vector;
function getAircraftFromRadarData (RadarData, string) : Aircraft;
function newAircraft (string) : Aircraft;
function isAircraftNull (Aircraft) : boolean;

inputoutput S_airplanes : RadarData;
inputoutput S_comingplanes : RadarData;

  % handle messages
  var
      airplanes      : RadarData,
      comingplanes  : RadarData,
      msg           : Message,
      src           : string,
      aircraft       : Aircraft
  in
    airplanes := ?S_airplanes;
    comingplanes := ?S_comingplanes;
    trap T in
      loop
        if(isListAEmpty()) then
          exit T;
        else
          msg := dequeueFromA ();
          if isReportMessage(msg) then
            src := getMessageSrc(msg);
            aircraft := getAircraftFromRadarData (airplanes,src);
            if not isAircraftNull(aircraft) then
              call setAircraftPosition (aircraft) (getMessagePosition(msg));
              call setAircraftVelocity (aircraft) (getMessageVelocity(msg));
            end if;
          elsif isHandoffMessage(msg) then
              % assume that is an incomming handoff (because of queue A)
              aircraft := newAircraft (getMessageReference(msg));
              call setAircraftVelocity (aircraft) (getMessageVelocity(msg));
              call addAircraftToRadarData (comingplanes) (aircraft);
          end if
        end if;
      %  if(not isListBEmpty()) then
      %
      %  end if;
      pause;
      end loop
    end trap;
```

```
    pause;
    emit S_airplanes(airplanes);
    emit S_comingplanes(comingplanes);
  end var;
end module


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% module for Syncronization of radardata and intern airplanes
%  Assumption: S_airplanes and S_comingplanes are emited before
module SyncRadar:
type Vector;
type Aircraft;
type RadarData;
type Message;

 %% A declaration of each procedure implemented in AtcControlInterface.java
procedure makeVector (Vector) (integer, integer, integer);
procedure updateAircraftPositionInRadarData (RadarData) (Aircraft);
procedure addAircraftToRadarData (RadarData) (Aircraft);
procedure removeAircraftFromRadarData (RadarData) (Aircraft);
procedure removeRadarDataIteratorItem () ();
procedure addMessageTargetItem()(Aircraft);
procedure removeMessageTargetItem()(Aircraft);
procedure setAircraftVelocity (Aircraft) (Vector);
procedure setAircraftColour (Aircraft) (Vector);

%% A declaration of each function implemented in AtcControlInterface.java
function initialiseRadarDataIterator (RadarData) : integer;
function nextRadarDataIteratorItem () : Aircraft;
function radarDataContainsAircraft(RadarData, Aircraft) : boolean;
function getAircraftFromRadarData (RadarData, string) : Aircraft;
function isAircraftNull (Aircraft) : boolean;
function getAircraftID (Aircraft) : string;
function getAircraftPosition (Aircraft) : Vector;
function getAircraftVelocity (Aircraft) : Vector;
function getAircraftColour (Aircraft) : Vector;
function hasNextRadarDataIteratorItem () : boolean;

input radar : RadarData;
inputoutput S_airplanes : RadarData;
inputoutput S_comingplanes : RadarData;

  % Update newcomers or leaved planes
  var
      airplanes      : RadarData,
      comingplanes  : RadarData,
      msg           : Message,
      src           : string,
      number         : integer,
      comingairc    : Aircraft,
      aircraft       : Aircraft,
      col           : Vector
  in
    airplanes := ?S_airplanes;
    comingplanes := ?S_comingplanes;


    % Go through all planes in our my airplanes and look if there are in radardata as well
    number := initialiseRadarDataIterator (airplanes);
    repeat number times
        if hasNextRadarDataIteratorItem() then
          aircraft := nextRadarDataIteratorItem ();
          % is this not in radardata?
          if not radarDataContainsAircraft(?radar, aircraft) then
            % through airplane out of local radar and Message Control
```

```
            call removeRadarDataIteratorItem()();
            call removeMessageTargetItem ()(aircraft);
          else % reset colour
            call makeVector(col)(0,200,0);
            call setAircraftColour(aircraft)(col);
          end if
        end if;
        pause;
    end repeat;

    % Go through all planes in radardata and look if there are in my airplanes var
    number := initialiseRadarDataIterator (?radar);
    repeat number times
        if hasNextRadarDataIteratorItem() then
          aircraft := nextRadarDataIteratorItem ();
          % is this in my airplanes?
          if radarDataContainsAircraft(airplanes, aircraft) then
            % update pos
            call updateAircraftPositionInRadarData (airplanes) (aircraft);
          else
            % add aircraft to local radardata and to Messageinterface
            call addMessageTargetItem ()(aircraft);
            % look for handoffmessage
            comingairc := getAircraftFromRadarData (comingplanes,getAircraftID(aircraft));
            if not isAircraftNull(comingairc) then % found
              call setAircraftVelocity (aircraft) (getAircraftVelocity(comingairc));
              call removeAircraftFromRadarData (comingplanes) (comingairc);
            end if;
            call addAircraftToRadarData (airplanes) (aircraft);
          end if
        end if;
        pause;
    end repeat;

    pause;
    emit S_airplanes(airplanes);
    emit S_comingplanes(comingplanes);
  end var;
end module

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% module for predicting the positions of the airplanes after 5 sec
%  Assumption: S_airplanes and S_comingplanes are emited before
module PredictPositions:
type Vector;
type Aircraft;
type RadarData;
type Message;

 %% A declaration of each procedure implemented in AtcControlInterface.java
procedure setAircraftVelocity (Aircraft) (Vector);
procedure setAircraftPosition (Aircraft) (Vector);
procedure setAircraftColour (Aircraft) (Vector);

%% A declaration of each function implemented in AtcControlInterface.java
function initialiseRadarDataIterator (RadarData) : integer;
function nextRadarDataIteratorItem () : Aircraft;
function getAircraftPosition (Aircraft) : Vector;
function getAircraftVelocity (Aircraft) : Vector;
function getAircraftColour (Aircraft) : Vector;
function hasNextRadarDataIteratorItem () : boolean;
function vectorAddition (Vector, Vector) : Vector;
function vectorMultiplication (Vector, integer) : Vector;
function vectorDivision (Vector, integer) : Vector;

input radar : RadarData;
```

```
inputoutput S_airplanes : RadarData;

  % Update newcomers or leaved planes
  var
      airplanes      : RadarData,
      pos           : Vector,
      number         : integer,
      aircraft       : Aircraft,
      col            : Vector
  in
    airplanes := ?S_airplanes;

    % Go through all planes in our my airplanes and predict position
    number := initialiseRadarDataIterator (airplanes);
    repeat number times
        if hasNextRadarDataIteratorItem() then
          aircraft := nextRadarDataIteratorItem ();
          pos := getAircraftPosition (aircraft);
          % velocity is calculated by vector times seconds
          pos := vectorAddition(pos,vectorMultiplication(getAircraftVelocity(aircraft),5));
          call setAircraftPosition (aircraft) (pos);
          col := vectorDivision(vectorMultiplication(getAircraftColour (aircraft),8),10);
          call setAircraftColour (aircraft) (col);
        end if;
        pause;
    end repeat;
    pause;
    emit S_airplanes(airplanes);
  end var;
end module

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
module AtcControl :

type Vector;
type Aircraft;
type RadarData;
type Message;


input radar:RadarData;
input inAirspaceMessage:Message;
input inControlMessage:Message;
output display:RadarData;
output outControlMessage:Message;
output outAirspaceMessage:Message;

%% A declaration of each procedure implemented in AtcControlInterface.java
procedure enqueueOnA ()(Message);
procedure enqueueOnB ()(Message);

  var airplanes      : RadarData,
      comingplanes  : RadarData,
      msg           : Message
  in
    signal S_airplanes     : RadarData,
           S_comingplanes  : RadarData
    in
      every radar do
          % Handle Messages
          run MessageHandler;

          % Synchronize radar and airplanes
          run SyncRadar;

          % get results
```

```
          airplanes := ?S_airplanes;
          comingplanes := ?S_comingplanes;
          emit display(airplanes);

          % update airplanes every 5 seconds
          every 500 tick do
            run PredictPositions;
            airplanes := ?S_airplanes;
            emit display(airplanes);
          end every

      end every
    end signal;
  end var
  ||
  every inAirspaceMessage do
    call enqueueOnA () (?inAirspaceMessage);
    emit outControlMessage(?inAirspaceMessage)
  end every
  ||
  every inControlMessage do
    emit outAirspaceMessage(?inControlMessage)
  end every

end module

```

Stage 4

At first I fixed this timing problem, that the display is not updated in equidistant time periods. Therefore I introduce a new local signal `S_timing`. This signal counts the `pause` commands in the code. Before displaying I wait now only $500 -$ `?S\_timing` ticks. To avoid that the data in `lastdisplay` becomes change through further calculations I have to wait a certain amount of time before continuing processing. (I choose 200 (2 seconds))

1 I added the necessary functionality to a new Module `CheckSpace`, where each plane were compared with all other planes using the function `isTooClosed`. If a plane has no interactions than it is drawn green else red.

2 I added the outgoing Handoff messages to queue B. Now there are 2 new points of processing necessary. If plane leaves than the appropriate Handoff message should be deleted from the queue B if exists. If an airplane is about to enter an neighbouring airspace than check if there is a Handoff message. My definition of "about to enter" is if the plane will leave the controlled airspace in the next 30 seconds. For this 2 new tasks I wrote 2 new modules `RemoveHandoffMessagFromB` and `checkLeaveHandoff`. The latter module is called from `CheckSpace` if the plane is about to enter an neigbouring airspace. if there is no handoff message the signal `S_orange` is true.

AtcControl.strl.Stage4

```
% Defines the module for the outermost control structure for the ATC
% Reactive Computer System.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% module for handling inAirspace Messages in the queue A
module MessageHandler:
type Vector;
type Aircraft;
type RadarData;
type Message;
```

```
procedure enqueueOnA ()(Message);
procedure setAircraftPosition (Aircraft) (Vector);
procedure setAircraftVelocity (Aircraft) (Vector);
procedure setAircraftColour (Aircraft) (Vector);
procedure addAircraftToRadarData (RadarData) (Aircraft);
function dequeueFromA () : Message;
function isListAEmpty () : boolean;
function isHandoffMessage (Message) : boolean;
function isReportMessage (Message) : boolean;
function getMessageSrc (Message) : string;
function getMessageTgt (Message) : string;
function getMessageReference (Message) : string;
function getMessageTime (Message) : integer;
function getMessagePosition (Message) : Vector;
function getMessageVelocity (Message) : Vector;
function getAircraftFromRadarData (RadarData, string) : Aircraft;
function newAircraft (string) : Aircraft;
function isAircraftNull (Aircraft) : boolean;

inputoutput S_airplanes : RadarData;
inputoutput S_comingplanes : RadarData;

  % handle messages
  var
      airplanes      : RadarData,
      comingplanes  : RadarData,
      msg           : Message,
      src           : string,
      aircraft       : Aircraft
  in
    present
      case S_airplanes do
        airplanes := ?S_airplanes;
      case  S_comingplanes do
        comingplanes := ?S_comingplanes;
    end present;
    trap T in
      loop
        if(isListAEmpty()) then
          exit T;
        else
          msg := dequeueFromA ();
          if isReportMessage(msg) then
            src := getMessageSrc(msg);
            aircraft := getAircraftFromRadarData (airplanes,src);
            if not isAircraftNull(aircraft) then
              call setAircraftPosition (aircraft) (getMessagePosition(msg));
              call setAircraftVelocity (aircraft) (getMessageVelocity(msg));
            end if;
          elsif isHandoffMessage(msg) then
              % assume that is an incomming handoff (because of queue A)
              aircraft := newAircraft (getMessageReference(msg));
              call setAircraftVelocity (aircraft) (getMessageVelocity(msg));
              call addAircraftToRadarData (comingplanes) (aircraft);
          end if
        end if;
      pause;
      end loop
    end trap;
    pause;
    emit S_airplanes(airplanes);
    emit S_comingplanes(comingplanes);
  end var;
end module

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% module for checking whether an aircaft is about to
% leave without handoff message
% about to leave means: plane will leave in 30 seconds
  %  Assumption: - S_aircraft, S_timing emitted
  %              - handoff message is the right one
module checkLeaveHandoff:
type Message;
type Aircraft;
type Vector;

procedure makeVector (Vector) (integer, integer, integer);
procedure enqueueOnB ()(Message);
function getAircraftID (Aircraft) : string;
function getAircraftPosition  (Aircraft) : Vector;
function getAircraftVelocity (Aircraft) : Vector;
function dequeueFromB () : Message;
function getMessageReference (Message) : string;
function sizeOfListB()   : integer;
function eq_string(string, string)   : boolean;
function vectorInside(Vector, Vector, Vector) : boolean;
function getAirspaceBounds() : Vector;
function vectorAddition (Vector, Vector) : Vector;
function vectorMultiplication (Vector, integer) : Vector;

input S_aircraft : Aircraft;
output S_orange : boolean;
inputoutput S_timing : integer;

  var msg   : Message,
      id    : string,
      pos   : Vector,
      nullv  : Vector,
      number: integer,
      timing: integer
  in
    timing := ?S_timing;
    id := getAircraftID(?S_aircraft);
    pos := getAircraftPosition(?S_aircraft);
    % check if plane is about to leave (30 seconds)
    % velocity is calculated by vector times seconds
    pos := vectorAddition(pos,vectorMultiplication(getAircraftVelocity(?S_aircraft),30));
    call makeVector(nullv)(0,0,0);
    % is in the Airspacebouunds?
    if not vectorInside(pos, nullv, getAirspaceBounds()) then
      number := sizeOfListB();
      trap T in
        repeat number times
          msg := dequeueFromB();
          call enqueueOnB()(msg);
          if eq_string(getMessageReference(msg),id) then
            % here should be checked if it is the right one, but to keep it simple
            %  we assume that is is the right
            emit S_orange(false);
            exit T;
          end if;
          pause;
          timing := timing + 1;
        end repeat;
        emit S_orange(true);
      end trap
    else
         emit S_orange(false);
    end if;
    pause;
    emit S_timing(timing + 1);
  end var
end module
```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% module for removing Handoff message of S_aircraft from queue B
  %  Assumption: S_airplane is emitted
  %              there is only on handoff message per aircraft (remove only first)
module removeHandoffMessageFromB:
type Aircraft;
type Message;

procedure enqueueOnB ()(Message);
function getAircraftID (Aircraft) : string;
function dequeueFromB () : Message;
function isListAEmpty () : boolean;
function isListBEmpty () : boolean;
function getMessageReference (Message) : string;
function sizeOfListB()   : integer;
function eq_string(string, string)   : boolean;

input S_aircraft : Aircraft;

  var msg : Message,
      id  : string,
      number : integer
  in
    id := getAircraftID(?S_aircraft);
    number := sizeOfListB();
    trap T in
      repeat number times
        msg := dequeueFromB();
        if eq_string(getMessageReference(msg),id) then
          exit T;
        else
          call enqueueOnB()(msg);
        end if;
        pause;
      end repeat
    end trap;
    pause;
  end var
end module

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% module for Syncronization of radardata and intern airplanes
  %    (also remove Handoff message from queue B if planes leaves)
  %  Assumption: S_airplanes and S_comingplanes are emited before
module SyncRadar:
type Vector;
type Aircraft;
type RadarData;
type Message;

 %% A declaration of each procedure implemented in AtcControlInterface.java
procedure makeVector (Vector) (integer, integer, integer);
procedure updateAircraftPositionInRadarData (RadarData) (Aircraft);
procedure addAircraftToRadarData (RadarData) (Aircraft);
procedure removeAircraftFromRadarData (RadarData) (Aircraft);
procedure removeRadarDataIteratorItem () ();
procedure addMessageTargetItem()(Aircraft);
procedure removeMessageTargetItem()(Aircraft);
procedure setAircraftVelocity (Aircraft) (Vector);
procedure setAircraftColour (Aircraft) (Vector);

%% A declaration of each function implemented in AtcControlInterface.java
function initialiseRadarDataIterator (RadarData) : integer;
function nextRadarDataIteratorItem () : Aircraft;
```

```
function radarDataContainsAircraft(RadarData, Aircraft) : boolean;
function getAircraftFromRadarData (RadarData, string) : Aircraft;
function isAircraftNull (Aircraft) : boolean;
function getAircraftID (Aircraft) : string;
function getAircraftPosition (Aircraft) : Vector;
function getAircraftVelocity (Aircraft) : Vector;
function getAircraftColour (Aircraft) : Vector;
function hasNextRadarDataIteratorItem () : boolean;

input radar : RadarData;
inputoutput S_airplanes : RadarData;
inputoutput S_comingplanes : RadarData;

  % Update newcomers or leaved planes
  var
      airplanes      : RadarData,
      comingplanes  : RadarData,
      msg           : Message,
      src           : string,
      number         : integer,
      comingairc    : Aircraft,
      aircraft       : Aircraft
  in
    airplanes := ?S_airplanes;
    comingplanes := ?S_comingplanes;
    pause;

    % Go through all planes in our my airplanes and look if there are in radardata as well
    number := initialiseRadarDataIterator (airplanes);
    repeat number times
        if hasNextRadarDataIteratorItem() then
          aircraft := nextRadarDataIteratorItem ();
          % is this not in radardata?
          if not radarDataContainsAircraft(?radar, aircraft) then
            % through airplane out of local radar and Message Control
            call removeRadarDataIteratorItem()();
            call removeMessageTargetItem ()(aircraft);
            % remove Handoff message from queue B
            signal S_aircraft: Aircraft in
              emit S_aircraft(aircraft);
              run removeHandoffMessageFromB;
            end signal
          end if
        end if;
        pause;
    end repeat;

    % Go through all planes in radardata and look if there are in my airplanes var
    number := initialiseRadarDataIterator (?radar);
    repeat number times
        if hasNextRadarDataIteratorItem() then
          aircraft := nextRadarDataIteratorItem ();
          % is this in my airplanes?
          if radarDataContainsAircraft(airplanes, aircraft) then
            % update pos
            call updateAircraftPositionInRadarData (airplanes) (aircraft);
          else
            % add aircraft to local radardata and to Messageinterface
            call addMessageTargetItem ()(aircraft);
            % look for handoffmessage
            comingairc := getAircraftFromRadarData (comingplanes,getAircraftID(aircraft));
            if not isAircraftNull(comingairc) then % found
              call setAircraftVelocity (aircraft) (getAircraftVelocity(comingairc));
              call removeAircraftFromRadarData (comingplanes) (comingairc);
            end if;
            call addAircraftToRadarData (airplanes) (aircraft);
          end if
```

```
        end if;
        pause;
    end repeat;

    pause;
    emit S_airplanes(airplanes);
    emit S_comingplanes(comingplanes);
  end var;
end module

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% module for predicting the positions of the airplanes after 5 sec
%  Assumption: S_airplanes, S_timing is emited before
module PredictPositions:
type Vector;
type Aircraft;
type RadarData;
type Message;

 %% A declaration of each procedure implemented in AtcControlInterface.java
procedure setAircraftVelocity (Aircraft) (Vector);
procedure setAircraftPosition (Aircraft) (Vector);
procedure setAircraftColour (Aircraft) (Vector);

%% A declaration of each function implemented in AtcControlInterface.java
function initialiseRadarDataIterator (RadarData) : integer;
function nextRadarDataIteratorItem () : Aircraft;
function getAircraftPosition (Aircraft) : Vector;
function getAircraftVelocity (Aircraft) : Vector;
function getAircraftColour (Aircraft) : Vector;
function hasNextRadarDataIteratorItem () : boolean;
function vectorAddition (Vector, Vector) : Vector;
function vectorMultiplication (Vector, integer) : Vector;
function vectorDivision (Vector, integer) : Vector;

input radar : RadarData;
inputoutput S_airplanes : RadarData;
inputoutput S_timing : integer;

  % Update newcomers or leaved planes
  var
      airplanes      : RadarData,
      pos           : Vector,
      number         : integer,
      aircraft       : Aircraft,
      timing        : integer
  in
    timing := ?S_timing;
    airplanes := ?S_airplanes;

    % Go through all planes in our my airplanes and predict position
    number := initialiseRadarDataIterator (airplanes);
    repeat number times
        if hasNextRadarDataIteratorItem() then
          aircraft := nextRadarDataIteratorItem ();
          pos := getAircraftPosition (aircraft);
          % velocity is calculated by vector times seconds
          pos := vectorAddition(pos,vectorMultiplication(getAircraftVelocity(aircraft),5));
          call setAircraftPosition (aircraft) (pos);
        end if;
        pause;
        timing := timing + 1;
    end repeat;
    pause;
    emit S_airplanes(airplanes);
    emit S_timing(timing + 1);
```

```
  end var;
end module

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% module for checking the space constrains of each the airplane
%  Assumption: S_airplanes, S_timing is emited before
module CheckSpace:
type Vector;
type Aircraft;
type RadarData;
type Message;

 %% A declaration of each procedure implemented in AtcControlInterface.java
procedure makeVector (Vector) (integer, integer, integer);
procedure setAircraftColour (Aircraft) (Vector);

%% A declaration of each function implemented in AtcControlInterface.java
function initialiseRadarDataIterator (RadarData) : integer;
function nextRadarDataIteratorItem () : Aircraft;
function getAircraftPosition (Aircraft) : Vector;
function getAircraftVelocity (Aircraft) : Vector;
function getAircraftColour (Aircraft) : Vector;
function hasNextRadarDataIteratorItem () : boolean;
function isTooClose (Vector, Vector) : boolean;
function willCrossing (Vector, Vector, Vector, Vector) : boolean;

input radar : RadarData;
inputoutput S_airplanes : RadarData;
input S_normalcolour : Vector;
inputoutput S_timing : integer;

  var
      airplanes      : RadarData,
      number         : integer,
      counter        : integer,
      aircraft       : Aircraft,
      pos           : Vector,
      refaircraft    : Aircraft,
      refnumber      : integer,
      refpos        : Vector,
      hasdetect      : boolean,
      orange        : boolean,
      orangecol     : Vector,
      redcol        : Vector,
      timing        : integer
  in
    timing := ?S_timing;
    airplanes := ?S_airplanes;

    refnumber := 0;
    call makeVector(redcol)(255,0,0);
    call makeVector(orangecol)(255,150,0);

    % Go through all planes in our my airplanes and check with others
    trap T in
      loop
        number := initialiseRadarDataIterator (airplanes);
        pause;
        hasdetect := false;
        if refnumber = number then % all planes testet
          exit T;
        end if;
        % go to new reference  plane
        repeat refnumber times
          aircraft := nextRadarDataIteratorItem ();
          pause;
```

```
        end repeat;
        refaircraft := nextRadarDataIteratorItem ();
        refpos := getAircraftPosition (refaircraft);

        % look for handoff message if it is about to leave
        signal S_aircraft : Aircraft,
               S_orange : boolean
        in
          emit S_aircraft(refaircraft);
          emit S_timing (timing);
          run checkLeaveHandoff;
          timing := ?S_timing;
          orange := ?S_orange;
          if orange then
            call setAircraftColour (refaircraft) (orangecol);
          end if
        end signal;

        refnumber := refnumber + 1;

        % compare refaircraft with all others
        number := initialiseRadarDataIterator (airplanes);
        counter := 0;
        repeat number times
            counter := counter + 1;
            if hasNextRadarDataIteratorItem() then
              aircraft := nextRadarDataIteratorItem ();
              if counter <> refnumber then
                pos := getAircraftPosition (aircraft);
                if isTooClose(refpos, pos) then
                  hasdetect := true;
                  call setAircraftColour (refaircraft) (redcol);
                end if
              end if
            end if;
            pause;
            timing := timing + 1;
        end repeat;
        if not hasdetect and not orange then % if this plane doesn't have any problems
          call setAircraftColour (refaircraft) (?S_normalcolour);
        end if
      end loop
    end trap;
    emit S_airplanes(airplanes);
    emit S_timing(timing);
  end var;
end module


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
module AtcControl :

type Vector;
type Aircraft;
type RadarData;
type Message;


input radar:RadarData;
input inAirspaceMessage:Message;
input inControlMessage:Message;
output display:RadarData;
output outControlMessage:Message;
output outAirspaceMessage:Message;

%% A declaration of each procedure implemented in AtcControlInterface.java
```

```
procedure makeVector (Vector) (integer, integer, integer);
procedure enqueueOnA ()(Message);
procedure enqueueOnB ()(Message);

%% A declaration of each function implemented in AtcControlInterface.java
function isHandoffMessage (Message) : boolean;
function isReportMessage (Message) : boolean;
function vectorAddition (Vector, Vector) : Vector;
function vectorSubtraction (Vector, Vector) : Vector;
function vectorMultiplication (Vector, integer) : Vector;
function vectorDivision (Vector, integer) : Vector;

  var airplanes      : RadarData,
      msg           : Message,
      col            : Vector
  in
    signal S_airplanes     : RadarData,
           S_comingplanes  : RadarData,
           S_normalcolour  : Vector,
           S_timing        : integer

    in
      every radar do
        % set colour for normal planes
        call makeVector(col)(0,200,0);
        emit S_normalcolour(col);
        % Handle InAirspaceMessages
        run MessageHandler;

        % Synchronize radar and airplanes
        %    (also remove Handoff message from queue B if planes leaves)
        run SyncRadar;

        % Check Space Constrains
        run CheckSpace;

        % get results
        airplanes := ?S_airplanes;
        emit display(airplanes); % radar data unchanged in terms of positions

        % update airplanes every 5 seconds
        loop
          await 200 tick; % let the interface time to get the displaydata
          emit S_timing(200);

          % set the normal colour for a plane more dark (intensity 90%)
          col := vectorDivision(vectorMultiplication(col,8),10);
          emit S_normalcolour(col);

          run PredictPositions;
          % Check Space Constrains and Handoff
          run CheckSpace;

          airplanes := ?S_airplanes;
          await (500 - ?S_timing - 1) tick; % -1 to get in the previous second
          emit display(airplanes); % display with predicted position
        end loop;
      end every
    end signal
  end var
  ||
  every inAirspaceMessage do
    call enqueueOnA () (?inAirspaceMessage);
    emit outControlMessage(?inAirspaceMessage)
  end every
  ||
  every inControlMessage do
```

```
    if isHandoffMessage(?inControlMessage) then
      call enqueueOnB ()(?inControlMessage);
    end if;
    emit outAirspaceMessage(?inControlMessage)
  end every

end module

```

Written Java Code:

part of AtcControlInterface.java

```
...
  /**
   * Test if vector v is in cube described by a and b.
   * */
  public static boolean vectorInside(Vector v, Vector a, Vector b){
    return v.inside(a,b);
  }

  /**
   * A method to check whether two vectors violate the space
   * constraints as defined by the
   * constants.Constants.SPACE_CONSTRAINT vector.  Please read the
   * documentation accompanying the definition of this constant to
   * find out what area is covered by the exclusion zone.
   *
   * @param pva position vector a, describes the location of an aircraft.
   *
   * @param pvb position vector b, describes the location of a
   * different aircraft.
   *
   * @return true if the position vector b is located inside position
   * vector a's exclusion zone, as defined by the
   * constants.Constants.SPACE_CONSTRAINT vector.
   */
  public static boolean isTooClose(Vector pva, Vector pvb){
      return pvb.inside(Vector.sub (pva, new Vector(constants.Constants.SPACE_CONSTRAINT)),
                        Vector.add (pva, new Vector(constants.Constants.SPACE_CONSTRAINT)));
  }

  /**
   * @return Bounds of airspaceSpace
   * */
  public static Vector getAirspaceBounds(){
    return new Vector(constants.Constants.BOUNDS);
  }

...

  /**
   * generate a new aircraft with the given ID.
   *
   * @param id the new ID for this aircraft
   */
  public static Aircraft newAircraft(String id){
    return new  Aircraft(id);
  }

...

  /**
   *  string equal dont work with my esterel and java!
   *  Below is the equality implementation.
   */
  public static boolean eq_string(String a, String b){
    return a.equals(b);
  }
```

```

...

```

Part 3

Stages

1. • incomming handoff messages are shown in the Message Control.
   • outcomming messages like "Alter course" are achieving an Airspace.
2. • I watched all the time the entries of the drop down list "Target". Each new plane was inserted an all leaving planes were removed.
3. • First I tested only the report messages by disabling the processing of the handoff messages. If I sended a request for report to one and all airplanes. The velocities were attached successfuly.
   • After that I tested the handoff messages. It worked and each plane had its velocity after this step.
   • Now the refresh after 5 seconds are to be tested. I desided to attach a more dark green to the predicted planes. It works but it was not really 5 seconds! The first step took about 6 or 7 and the further about 5 or 6.
4. • Tested the new timing idea with the pause counter. Now each step is exactly 5 seconds.
   • I set the space constraints to a larger value $(1000, 1000, 999)$ to have more cases of such an violation.
   • The planes became red but not green again. I modified the code and now each plane is compared with all other and then I can decide if plane have to be red or not.
   • To test the leaving I considered three cases:
     i) I sended no Handoff message and the planes near the border became orange.
     ii) I sended a Handoff message long time before the plane is going to leave the controled airspace. The Airplane didn't become orange at all.
     iii) I sended a Handoff message after the plane was showed orange. The Airplane switched its colour to green (if no collision) again.