CS4 Module: Embedded Software
Assessed Exercise (2003-04)
Air Traffic Control (ATC) System

2 February, 2004

Deadline: 5 p.m. Friday, 5^{th} March 2004

Introduction

This project involves part of the design, implementation, and testing of a simplified air traffic monitoring and control system (ATC).

The main function of the ATC is to track aircraft travelling through its airspace and ensure their safety by being able to command any aircraft to change its course or speed. In this case, safety translates to avoiding aircraft collisions by maintaining a minimum separation distance between any two aircraft (this is predefined for you and can be found in the file `constants/Constants.java`. For the purposes of this assignment, you can assume that, unless there is a danger that two aircraft will violate the minimum separation distance, they will not be commanded to change their direction or speed.

Background

Note: All Java-related documentation can be accessed from:

`docs/overview-tree.html`

The aim of the assignment is to design, implement and test the ATC controller. A simulation of the 3-D airspace, displaying entry (exit) of aircraft to (from) the airspace is available to you. The ATC module, which is referred to as `AtcControl`, should itself be implemented in Esterel, however, it is accessible through a set of programmed interfaces (which have been implemented in Java) as follows:

1. interface to the operator who interacts with the ATC controller; its functionality is as follows:
    - Controller Communication In (Out) - communication from (to) the controlling operator to (from) the airspace via the `AtcControl` module.
    - See: `docs/MessageControlInterface.html`
2. interface between the ATC controller and:
    - the airspace which it controls &
    - the neighbouring ATC centres;

    its functionality is as follows:

    - Airspace Communication In (Out) - communication from (to) the airspace to (from) the `AtcControl` module.
    - See: `docs/AtcControlInterface.html`
3. the `AtcControl` module's interface to the radar display, comprising:
    - a radar subsystem (Radar In), to handle the radar input which arrives every 15 seconds
    - a display subsystem (Display Out) to output the processed radar information
    - See: `docs/AtcControlInterface.html`

The following assumptions can be made:

- Radar information always contains aircraft IDs and positions.
- Messages always reach their intended targets.
- Aircraft always obey orders.

Operators have no direct control over the airspace once the simulation is started.

ATC Controller Inputs

The inputs to the ATC controller (`AtcControl` module) can (simultaneously) arrive from:

1. the radar subsystem (3 above)
2. the airspace communications interface (2 above)
3. the operator interface (1 above)

Input type details can be found in the relevant file's documentation.

ATC Outputs

The ATC controller's outputs are to:

1. the airspace communications subsystem (2 above)
2. the operator interface (1 above)
3. the display subsystem (3 above)

Output type details can be found in the relevant file's documentation.

ATC Functionality

The functionality of the ATC is explained in the four stages of the Esterel implementation (see below).

Requirements

The exercise consists of three parts:

Part 1 Design the ATC Controller using Statecharts.

Hand-in: the design on paper.

Part 2 The functionality of the ATC Controller is to be implemented in Esterel. Note that you should **not** need to write any more Java code than what is already provided/outlined. If you feel you do, you are probably not using the full facilities of Esterel. This may carry a penalty unless you provide a valid justification.

This part is divided into four stages, to help break down the task:

Stage 1 - found in `AtcControl.strl.Stage1`. In this stage, you are simply required to establish contact between the `AtcControl` module and the inputs/outputs of the system i.e. get `AtcControl` module talking to the programmed interfaces so that you can handle inputs and outputs. No processing of inputs is required. Because you haven't identified any of the aircraft in your airspace, the ATC Controller can only output messages to the airspace by *broadcasting*.

Relevant java docs: `docs/AtcControlInterface.html`

Stage 2 - found in `AtcControl.strl.Stage2`. In this stage you need to glean aircraft *id* information from the radar data and use this to *update* the Message Control Interface. As aircraft enter the airspace, their IDs need to be added to the Message Control centre and as they leave, they need to be removed.

Additional relevant java docs:

- `docs/MessageControlInterface.html`
- `docs/Aircraft.html`
- `docs/RadarData.html`

Stage 3 - found in `AtcControl.strl.Stage3`. The displayed radar information currently contains no velocities. You now need to read and understand messages that deal with velocities and update the velocity information of aircraft accordingly. Report messages are used by aircraft to send their current coordinates and velocity to the ATC centre. Conversely, the ATC Centre sends instructive messages to the aircraft. Hand-off messages indicating where and when aircraft will appear and what their velocities will be are sent by neighbouring ATC centres. By the end of this stage, each aircraft displayed on the radar screen should always have its associated velocity. The radar display output should be altered so that the screen is updated every 5 seconds, instead of when radar data arrives (every 15 seconds).

Details of ATC messages can be found in: `docs/Message.html`

Stage 4 - found in `AtcControl.strl.Stage4`. This final programming stage needs to extend the Esterel specification further to implement the warnings that the ATC controller needs to provide. In this implementation warnings will be provided by manipulating the colours of planes. Two main warnings are required:

1. If two aircraft are too close then the offending aircraft must be highlighted for the operator to take action. Aircraft should be displayed in **RED** when space constraints, defined by `constants.Constants.SPACE_CONSTRAINT` vector, have been violated. To keep it simple we shall assume that the vector, `SPACE_CONSTRAINT`, defines an area that may contain a single aircraft at its centre. If another aircraft enters this safety zone then the alarm should go for both aircraft i.e. colours of both aircraft must be changed to RED to alert the operator. This will require you to fill in an implementation for the method `isTooClose( )` which is declared in the class `AtcControlInterface.java`.
2. Prevent aircraft from entering neighbouring airspace without a relevant hand-off message. If an aircraft is about to enter a neighbouring airspace and a hand-off message for that aircraft has not been sent to that airspace then the aircraft should be displayed orange. Defining *about to enter* is up to you, but must be explained clearly.

Only a file called `AtcControl.strl` will actually produce the correct Java implementation. So first copy the Esterel file you have been working on to `AtcControl.strl`. After you have completed one stage, you could copy it across to the next one and modify it to reflect the requirements of the latter, and so on. For a description of the functions and procedures defined in the Esterel skeletons, you can look at their implementations in the file, `AtcControlAppli.java`.

Hand-in:

1. A **hard copy** of your solutions *must* be delivered to the ITO by the deadline, with your name and matric number clearly marked.
2. Your Esterel sources for each of the above stages (or as far as you get) and `AtcControlInterface.java`, should be e-mailed to me "**dka@inf.ed.ac.uk**" as a **gzipped tar** file. Please also send me any other Java files you may have modified. Please mark your e-mails: **Embedded Software Submission**.

Part 3 Outline the tests you have performed to check the functionality of your system. Also, report on the results of testing your system under different operating conditions - e.g., by changing the amount of I/O traffic, the number of planes in the airspace at any time. You can do this by changing the values of appropriate constants in the `constants/Constants.java` file (`MAX_PLANES, PROBABILITY_NORTH, PROBABILITY_EAST`, etc.).

Handin: Include a **hard copy of your test report** with your submissions in **Part 2**.

The Components of the atc.tar.gz Archive

Here are the components of the `atc.tar.gz` archive:

1. `env_setup.sh` - a small shell script that sets up your environment so that the Java programs will compile and run correctly.

2. Java source files that can be split into several categories as follows:

   (a) The Java files containing classes which you will use in your Esterel specification and which have all been implemented for you (remember, procedures in Esterel are defined in the host language, which is Java in our case, and can be called from Esterel using the `call` statement):

   - `Aircraft.java`
   - `Message.java`
   - `RadarData.java`
   - `Vector.java`

   Refer to the html versions of the named files in `"docs/"` for detailed documentation. Note that you do not need to understand the actual implementation of these classes, it is sufficient to know their functionality so that you can simply `call` them from within Esterel as appropriate.

   (b) The Java files that are used by the Esterel specification when it has been compiled into a java class:

   - `AtcControlAppli.java` - you should not edit this
   - `AtcControlInterface.java` - all of this has been implemented for you, except for one or two methods for which you need to fill in an implementation - these are detailed in the documentation for the relevant stage of the assignment.

   (c) The Java files that are used by the `AtcControlInterface.java` class, but are not directly manipulated in the Esterel specification:

    - RadarDataIterator.java

   (d) The Java files that are used by the `AtcControlAppli` class to link in with the provided GUIs and airspace model in the `atc.jar` package:

    - `AirspaceInterface.java`
    - `MessageControlInterface.java`
    - `RadarDisplayInterface.java`

   (e) The `Constants` class that has a package to itself, and is, therefore, located in the relevant subdirectory. This class contains constants that may be altered as you wish (especially for testing purposes in **Part 2**, Stage 4):

    - `constants/Constants.java`

3. Skeleton Esterel specifications - these contain all the inputs, functions and procedures you will need to complete the exercises. At present, the outlines for all the stages are identical, except for the description at the top of each file which describes what needs to be accomplished for each stage:

   (a) AtcControl.strl.Stage1

   (b) AtcControl.strl.Stage2

   (c) AtcControl.strl.Stage3

   (d) AtcControl.strl.Stage4

4. `atc.jar` - the package that contains all the compiled java classes that control the environment to which the Esterel specification is linked. You DO NOT need to explore the contents of this directory in order to complete your assignment. If you feel you do, then your approach to the problem is probably more complicated than is necessary!

5. `docs/` - the directory that contains html documentation for the `.java` files in the archive. The documentation is automatically generated from the Java files in this directory, so where you read the information on each java file is a matter of preference. You will at least need to read the documentation accompanying the classes that define the types you will use in your Esterel specification and the documentation for the `AtcControlInterface` class.

6. `images/` - the directory that contains gif files used by the Java GUIs.

7. `classes/` - the directory is empty but will be used to contain the compiled java files as you progress through the practical.

Executing Your Esterel Programs

This section explains how you should set up the environment and compile your Esterel programs.

Before you try compiling or executing any Java command, you must execute the shell script provided to setup the environment. You must be in the `atc` directory when you run this script and when you execute the `esterel`, `javac`, and `java` commands.

So, try:

```
>:  .  env_setup.sh
```

Compiling an Esterel specification - Only a file called `AtcControl.strl` will actually produce the correct java implementation. So first copy the Esterel file you have been working on to `AtcControl.strl`.

```
>:  cp AtcControl.strl.StageN AtcControl.strl
```

Next compile the Esterel program to a `.java` file.

```
>:  esterel -Ljava AtcControl.strl
```

Now compile the java files to produce the required `.class` files in the `classes/` subdirectory. If you have not already done so, you will first need to compile the file, `constants/Constants.java`.

```
>:  javac -d classes constants/Constants.java
>:  javac -d classes *.java
```

You are now ready to test your implementation. You should not change directory because Java will not be able to find the necessary images otherwise.

>: java AtcControlAppli

You should get three windows: **Radar Display**, **Message Control**, and **Control Panel**. The three buttons stand for *play*, *step*, and *stop*. They start the simulation, increment simulation time by one, and stop the simulation, respectively.