Airport Traffic Control - Software Components Assignment

Jason Giancono
16065985

May 24, 2015

Contents

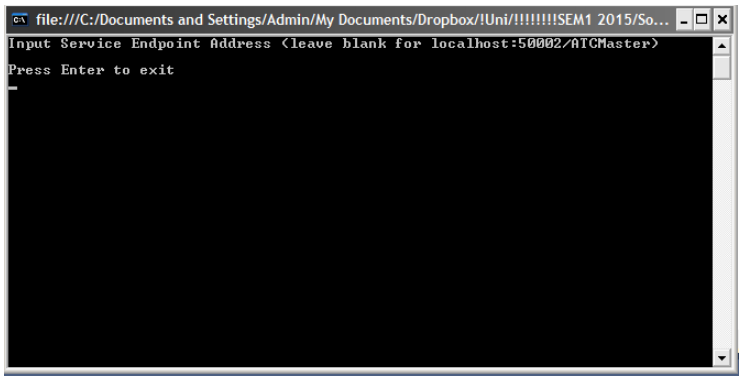


Figure 1: ATCMaster prompt screen

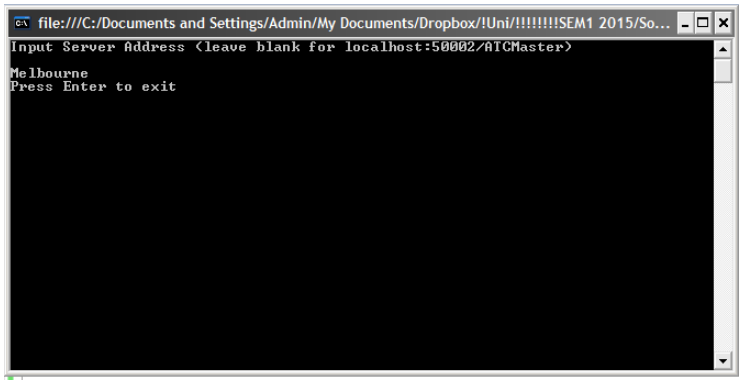


Figure 2: ATCSlave prompt screen

1 Running the Application

In order to get a working system up and running, complete the below steps

1.1 Start ATCMaster

Either open the ATCMaster solution and press F5 or go into

`ATCMaster/bin/Debug/ATCMaster.exe`

A command window will open. It will prompt you to enter an address to bind the service to. You can either enter an address or just press enter and it binds to the default address.

1.2 Start the ATCSlaves

You now need to start the slaves in the directory of

`ATCSlave/bin/Debug/ATCSlave.exe`

You will need to open the amount of slaves that the ATCMaster has airports, in the ATCDatabase.dll provided that number is 4. A command window will open

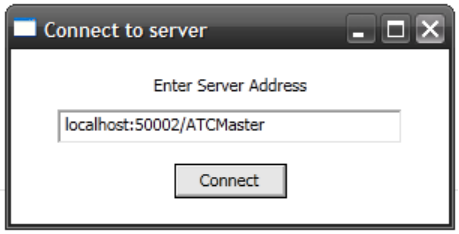


Figure 3: ATC WPF connect prompt

and prompt for a server address, enter the address you bound the ATCMaster instance to above. If you just used the default for ATCMaster you can use the default for these (assuming the slaves are on the same host. Once you enter the address it will connect and then display the Airport name that it has been allocated.

1.3 Start either Web or WPF GUI

You now need to either start the Web application or the WPF GUI application.

1.3.1 WPF Application

To start the WPF Application, go to the ATCPresentationGUI project and press F5 or go to

`ATCPresentationGUI/bin/Debug/ATCPresentationGUI.exe`

Enter the address of the Master server in the dialog box, then you can navigate through the different airports using the listBox at the top and then view the Airplanes connected to that airport in the ListViews below.

1.3.2 ASP.NET Website

The run the web GUI, first make sure the address string in Web.config is set to the address of the Master server. Then load up the ATCWeb solution and hit F5. You will now be able to see a table with the airport names in one column and a button in the other. Below the table is a button that says Step 15 Min, push this to run a step or view the airplanes in an airport by clicking on the button.

2 Critical Self-Analysis

2.1 "Next Step" function

One of the main function of the distributed system is the NextStep function. The assignment specification requires that the Master does the calls to the Slave asynchronously using blocking calls so that it is done in parallel. Originally my NextStep function in the Slave returned the Airport object, however I discovered that this was creating bugs where a plane might be passed from one airport to another after that airport had been returned. This meant that the wrong

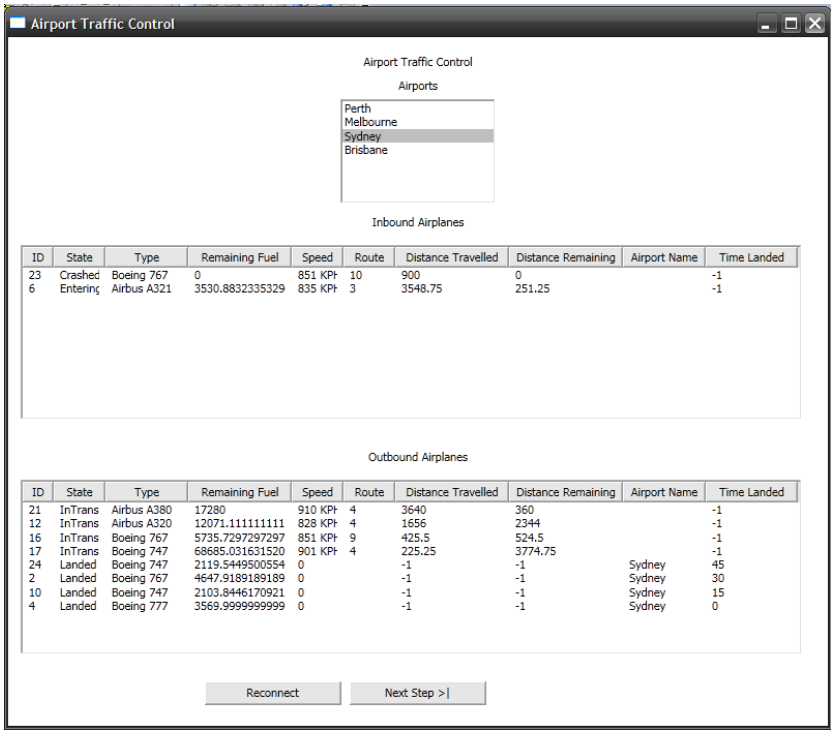


Figure 4: ATC WPF Application

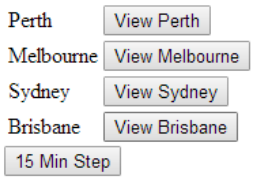


Figure 5: ATC ASP.NET Main Page

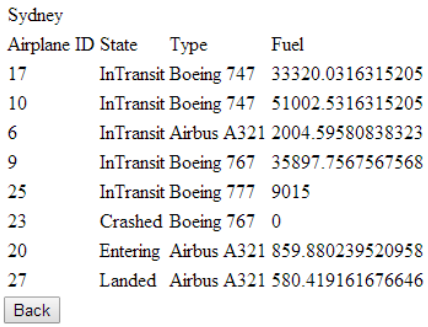


Figure 6: ATC ASP.NET Airplane View

airplanes were being displayed in GUI.

The solution for this was to do the parallel processing in two parts. The first would call NextStep in each of the slaves which would run in parallel and the next step was to connect to each slave again after they had all returned and requesting their Airport states. This added code complexity as I had to now to two sets of parallel calls but meant the airports displayed in the GUI was correct. Another solution could have been to get a copy of the Airport whenever you passed a plane and update a private field, however then you have the issue of managing the field between two threads and keeping it up to date so my solution seems simpler.

2.2 ASP.NET Web Page

My web front end is very simple. I am not experienced with this style of ASP.NET shown in the practicals (I have used MVC before though). As a result of this I decided to construct most of the page using ASP.NET objects (opposed to in the aspx file directly). Due to time constraints I could not flesh out the web page and it only meets the bare minimum of specs. If I had more time I would have implemented the following:

- Ajax requests
- Ability to step forward while viewing the planes
- Styling of the page to make it look less like the 90s

As suggested in the assignment specification, the ASP.NET server connects to the Master each time a request is made as ASP.NET is stateless. This could cause delays if the ASP.NET Server was far away from the Master server.

2.3 Object Passing Between Servers Considerations

In my design the only object the Presentation Applications (either Web or WPF) can get from the Master is the entire list of Airports. This is not the most efficient process however it is the simplest to implement.

If I were to optimise more for efficiency, I could add other operations for requesting individual airports, or just asking for airport names. Then the GUI/Web Service could just request the current airport that is selected rather than just getting all of them. This could make the WPF GUI less responsive, as you would have to do a WCF call every time you switched airports, where in my current version you only do a WCF call when you move a step. It would not have this issue with the ASP.NET server, in fact it would likely improve responsiveness.

Another issue with passing the whole Airport is that some of the data in an airport does not change, such as the AirRoutes or things like the airport name. These are still passed every update. I did not choose to use a method that addressed this redundancy as it would have added extra code complexity.

2.4 Building the Airports from ATCDatabase

I decided to have the Master construct all of the objects from the database and pass a single airport object to the slaves. I did this to reduce coupling between the database and the servers, by building them all entirely on the master the slave does not need to reference ATCDatabase at all. There is little downsides to this as the objects only need to built once and it doesn't take a prohibitive long time to build the entire list of airports. It also keeps the code simple as it is all in the one function and server.

Alternatively I could have had the slaves construct the AirRoute and Airplanes by accessing the database via WCF using functions on the master. This would have added to the code complexity.

2.5 Data Classes

The data classes have no functions except for the constructors (and the OnDeserialized function in the Airplane). The data classes are defined in ATCMaster, but they are used in each of the 4 projects. Because there is no other methods in the data classes they are easily passed around with WCF.

2.6 Separation of Airport States

The assignment specification suggested that the landed and queued airplanes could be combined however I decided to keep them in separated lists, so each airport has three lists of airplanes, landed, queued and departed. The reason for this is that it is much less complicated to combine two lists than it is to separate out airplanes in different states so that they can be processed differently each step.

2.7 Assumptions

In designing my systems I had to make some assumptions. These included

- At the beginning of the simulation, all airplanes have been landed for 0 minutes.
- When an airplane is landing, it does not use any fuel once it is at 0km. So if a plane is circling and it is selected, it will use 0 fuel that step. I originally had each plane burn an entire 15 minutes of fuel when it landed regardless of it's position but this ended up in a lot of crashes - some routes would crash on every single flight (because they were short flights and didn't have an extra 15 minutes of fuel).
- there is an assumption that a plane can't travel more than 300KM in 15 minutes. This has been the case with all the airplanes from the current ATCDatabase.dll but could change with others. The reason this is an assumption is because if a plane could do this then planes that are passed to another airport in a turn would have to be considered as a candidate for landing that turn (depending on their current fuel). This would make it difficult because it is possible that airport has already selected their landing plane and landed it by the time that airport is added.

2.8 Crash Circumstances

Running tests I have noticed several conditions that are met when planes crash. It is only ever on short trips (planes never crash on long routes such as those coming to/from Perth). Planes tend to crash when there are two or more queued who have little fuel left. Planes that travel short routes are more likely to crash as they have less excess fuel. One way to reduce the crash rate would be to alter the calculation on who to land first. Instead of just selecting the lowest fuel, you could select the plane with the lowest fuel/fuel consumption rate ratio.

2.9 Distributivity of the System

Currently the system has only been tested on a single machine. I have attempted to test it across computers in the labs but was unable to get around the firewall restrictions. If the system was to be distributed over a non-trivial network, I am unsure as to how it would perform. In my design I have prioritised simplicity over efficiency by only sending the entire list of airports from the master server, over a poor connection this could negatively affect the performance of the system.

The system itself is bottle-necked by the Master server. I do not think performance with many clients would be an issue unless many clients were trying to move to the next step, which is a synchronous operation.

3 Appendix

Please see the following pages for diagrams specified in the assignment specification.

1. High Level Design
2. UML Class Diagram
3. Sequence Diagram: Building Airport Object and Slave Connecting
4. Sequence Diagram: Clicking Next Step Button
5. Sequence Diagram: Viewing Airport from Web

Appendix 1: High Level Diagram

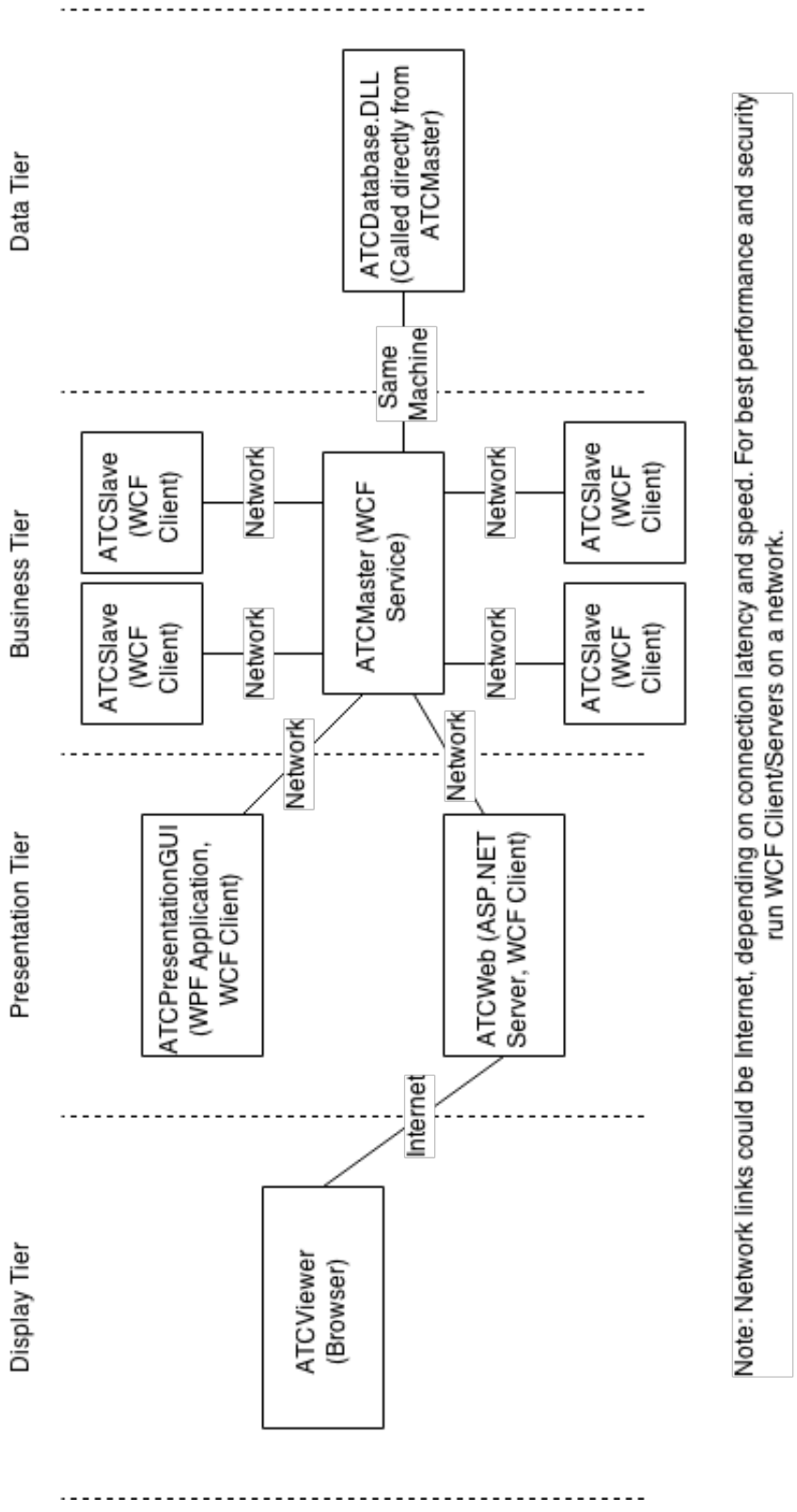
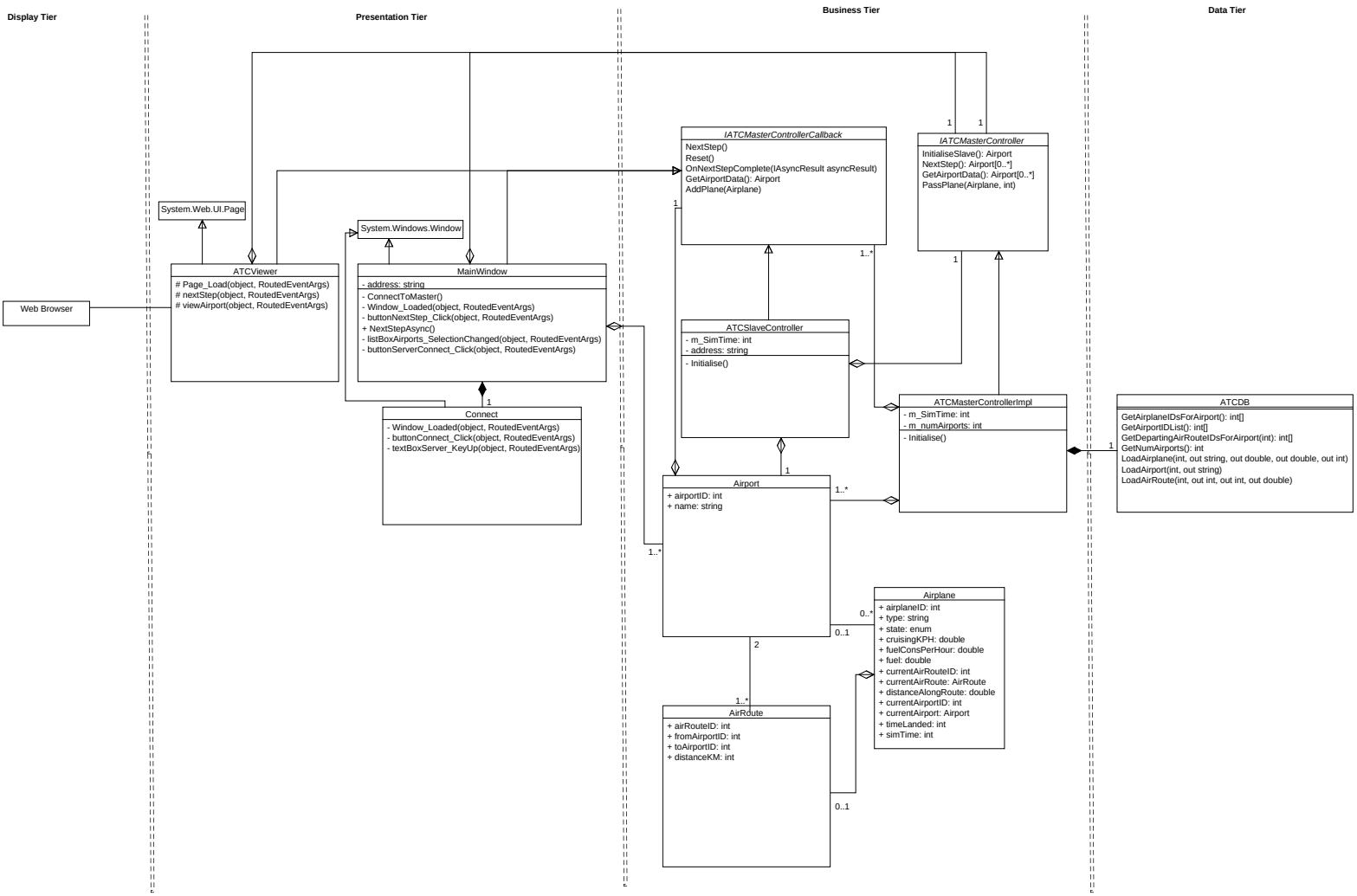
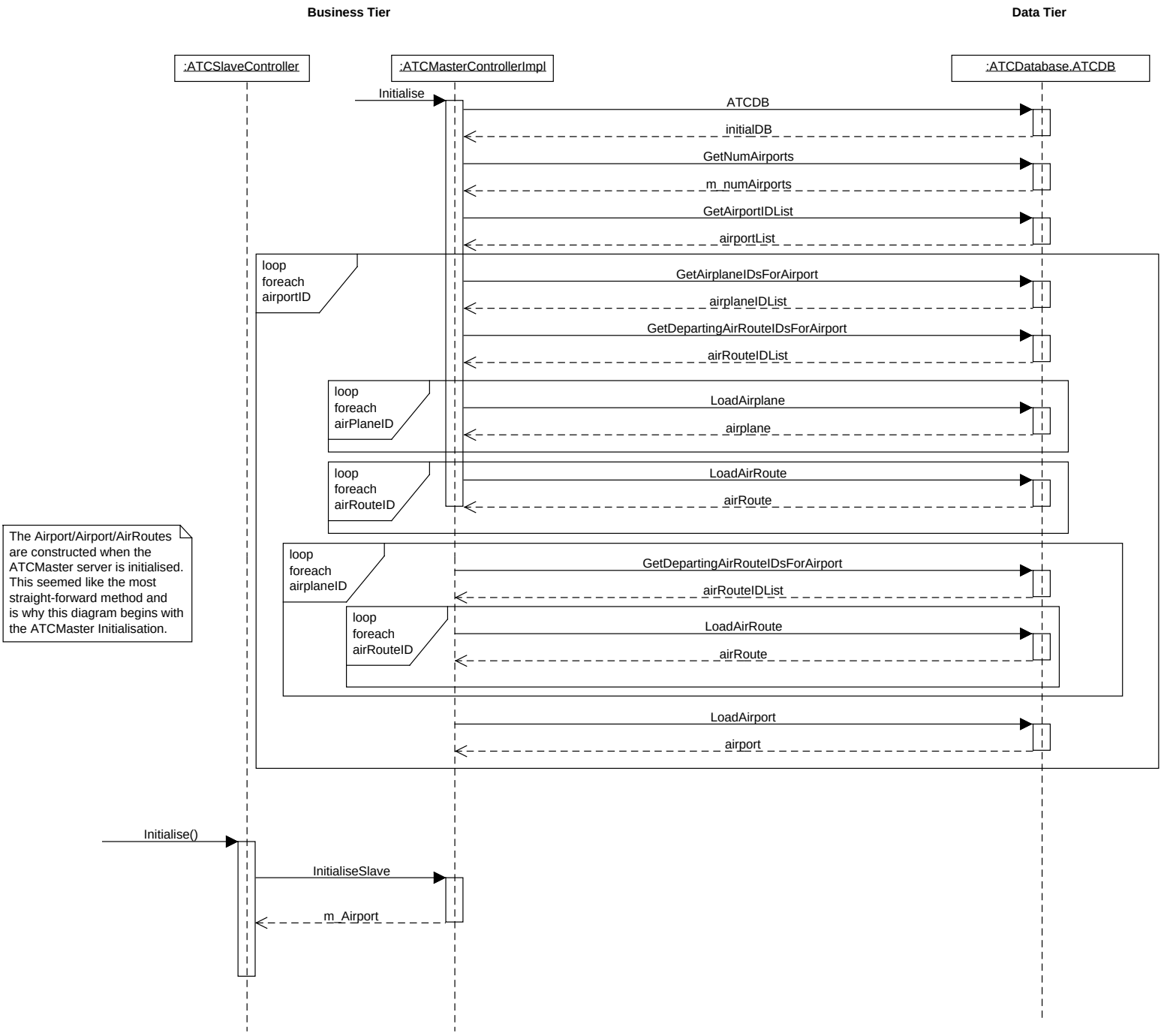


Figure 7: ATC Tier Diagram

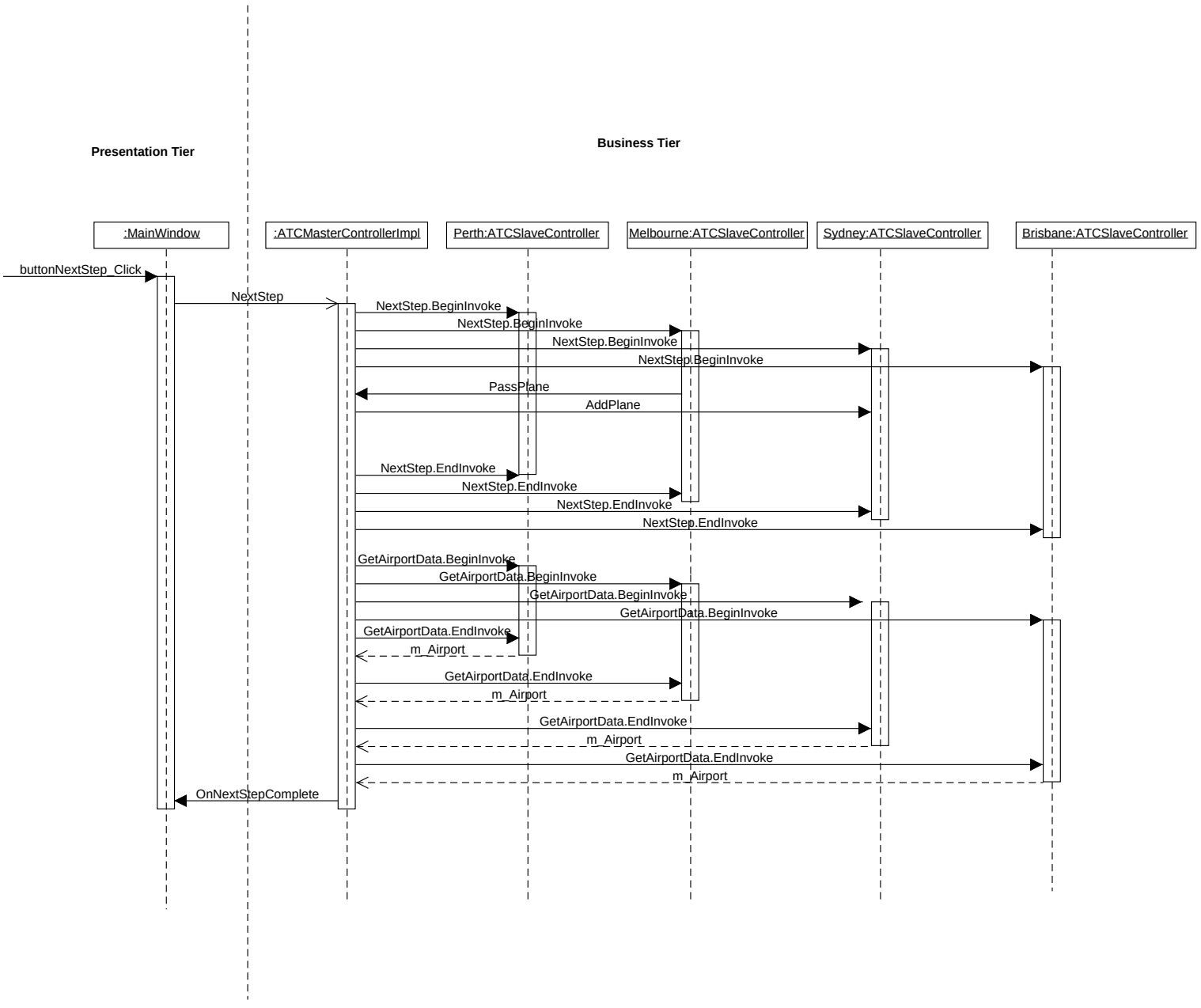
Appendix 2: UML Class Diagram



Appendix 3: Sequence Diagram: Building Airport Object and Slave Connecting



Appendix 4: Clicking Next Step Button



Appendix 5: Sequence Diagram: Viewing Airport from Web