CARE INO III

3D IN 2D PLANAR DISPLAY PROJECT

D2-1: ATC SIM AND EXPERIMENTATION TEST-BED
(LOT NO. 1, WP 2)

Reference : Edition 1 Effective Date 30/05/08

Authors          Organisation           Signature
Stephen Gaukrodger  Middlesex University
William Wong       Middlesex University
Alessandro Boccalatte  Space Application
                                        Services
Edition history

<table>
<thead>
<tr>
<th>Edition N°</th>
<th>Effective date or status</th>
<th>Author(s)</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30/05/2008</td>
<td>Gaukrodger, Wong, Boccalatte</td>
<td>Deliverable due.</td>
</tr>
</tbody>
</table>

Acknowledgements

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other members of the project team</td>
<td></td>
</tr>
<tr>
<td>Paola Amaldi</td>
<td>Middlesex University</td>
</tr>
<tr>
<td>Bob Fields</td>
<td>Middlesex University</td>
</tr>
<tr>
<td>Martin Loomes</td>
<td>Middlesex University</td>
</tr>
</tbody>
</table>
# Table of Contents

1. INTRODUCTION .................................................................................................................. 5  
   1.1 Simplified ATC Simulator Philosophy ........................................................................... 6  

2. ARCHITECTURE AND DESIGN ............................................................................................. 7  
   2.1 Capabilities .................................................................................................................. 7  
   2.2 The Development Roadmap ......................................................................................... 9  

3. PROBLEMS AND OPPORTUNITIES FOR IMPROVEMENT ........................................... 10  

4. IMPLEMENTED SOLUTIONS ................................................................................................. 12  
   4.1 Timeouts .................................................................................................................... 12  
   4.2 Keyboard Shortcuts ...................................................................................................... 12  
   4.3 Non Graphical Simulation ............................................................................................ 12  
   4.4 Scenario Editor Improvements ..................................................................................... 13  
   4.5 Smooth Path Following ............................................................................................... 13  
   4.6 The Pseudo-Pilot ......................................................................................................... 13  

5. CONCLUSION ...................................................................................................................... 14  

6. REFERENCES ......................................................................................................................... 14  

APPENDIX 1 THE SETTINGS.PROPERTIES FILE ................................................................. 15  

APPENDIX 2 ATC SIM SOFTWARE DOCUMENTATION ...................................................... 16
1. INTRODUCTION

This document D2-1 ATC Sim and Experimentation Test-bed closes the work for the development of the ATC simulator and experimentation test-bed of the 3D-in-2D Displays for ATC project, sponsored by EUROCONTROL CARE INO 3 programme.

The basic ATC simulator was developed to enable the project investigators to easily create dynamic air traffic scenarios and to use these scenarios to visualise and test the many 3D/2D HMI concepts that were expected to emerge during the project. The basic ATC simulator consists of the ATC Simulation Editor, an editing facility allowing the experimenters to create scenarios (typically including waypoints, fixes, weather and flying airplanes); a scenario database; a Pseudo-Pilot interface for ‘flying’ the aircraft within the simulation; the Simulator capable of running those ATC scenarios; with a compatible programmatic interface to the 3D/2D prototypes. Running a scenario requires interfacing with the different prototypes and providing real-time or fast-time simulated airspace data. Given the IRAB’s directive in December 2007 to change direction in this project, it was decided with EUROCONTROL that the Experimentation Test-bed would not be implemented. The Experimentation Test-bed was to have the functions needed to track user performance, responses and response times, to given changes in the ATC situation as part of experiments to evaluate the efficiency and effectiveness of the different interface and visualisation designs.

The change in direction required the project to focus its innovation and design efforts on SESAR, the Single European Sky ATM Research programme, in particular, in two areas: concepts for interaction and visualisations of SESAR concepts for the controller working positions, CWP, and for pilots in aircraft cockpits. While the SESAR Operational Concept has been defined (SESAR Consortium, 2007) the nature of work and ways of working has yet to be defined. Therefore rather than verifying and validating designs experimentally and therefore needing the experimentation test-bed, the emphasis was to rapidly develop innovative visualisation designs that could potentially support and potentially define the future SESAR working concepts for both the CWP and pilots. The basic ATC simulator would still be useful for this, and that it should be flexible enough to develop evaluation scenarios that can be used to run visualisations of and interactions with SESAR concepts.

In Year 1 (2007), the basic ATC Sim was used to provide ATC simulation scenarios for six prototypes in two evaluations and numerous demonstrations. Several key areas for improvement were identified. Most of these changes were small but made significant improvements to the usability of the system. However, one of these changes – smooth path
following – involved major changes to the code and the way data was stored and passed within the program. This change greatly simplified the program and made it possible to readily add an additional feature, the Pseudo-Pilot module.

The original version of the ATC Sim was described in D1.3 Architecture and Common Simulation Engine report. It included requirements, specifications and explanations of the interface that have largely remained unchanged. As such, this document will only describe the improvements that were made, the way the changes that were made, the rationale for those methods used and any problems that may arise as a result of these changes. In addition, a CD with the source code will be attached and Appendix 1 to this report will contain a full listing of the software documentation.

It should be noted that although this work will now be closed, we may, from time to time, find it necessary to make changes or additions to the code for the ATC Sim in order to improve it in necessary ways that we currently cannot anticipate. The most recent version of the ATC Sim software will be kept on site, and distributed as necessary.

1.1 Simplified ATC Simulator Philosophy

In Year 1, the project team chose to develop an easy-to-use tool for creating relatively simple air traffic scenarios for HMI testing and demonstration purposes. Special emphasis was placed on the evaluation of the Year 1 prototypes of the project, in particular, of the visuo-spatial aspects and expected benefits of innovative 3D-in-2D HMI concepts. For this reason, simplicity and ease of use, especially to connect with the prototypes, have been considered as important points.

While other existing, more sophisticated platforms such as AD4, ESCAPE, eDEP, would have provided much more realistic ATC simulation scenarios, such complexity and realism was considered to be beyond the scope of the early phases of the project. Instead, the focus of the ATC Sim is more to support the identification and evaluation of HMI aspects rather than to support operationally realistic assessments.
2. ARCHITECTURE AND DESIGN

The ATC Sim and Experimentation Test-bed comprises:

1. an editing tool, provided with an intuitive graphical interface, which supports the creation and editing of simple ATC scenarios, and

2. a simulation environment (simulation engine) capable to run the scenarios and to interface to the different 3D-in-2D HMI prototypes, feeding them with (near) real-time traffic data.

The following figure provides an overview of the testbed architecture, showing the different modules and the major data flows between them.

![Testbed Architecture Diagram]

Figure 1. Testbed architecture.

2.1 Capabilities

The system supports the scenario design process by means of a graphical user interface which allows the user to easily create basic ATC elements, such as fixes, waypoints, weather elements, and to define flight plans.
Figure 2. HMI of the simulator editor

The ATC editing environment supports a number of capabilities, allowing the user to:

- specify a geographic region containing the airspace of the scenario under creation;
- create a set of waypoints ("fixes");
- create aircraft flight plans, specifying information like the trajectory, the start time of the flight, the flight levels and the speed at which the aircraft will fly;
- create simple weather objects (e.g. representing cumulonimbi) defined as 3D volumes;
- verify the presence of potential events in the scenario, such as conflicts between airplanes (loss of separation) or airplanes crossing a weather object;
- play / preview the scenario in 2D
- insert textual comments and other auxiliary information for each scenario.

The created scenarios can be then performed by the ATC simulator, which generates the data of flying airplanes according to the flight plans specified in the scenarios, and dispatches those data to the HMI concept prototypes that are connected to it.

The data transmission from the ATC simulator to the HMI prototypes is realized by server-client communication scheme, based on the TCP-IP protocol, in which the simulator acts as a server, while the HMIs act as clients.
2.2 The Development Roadmap

The original expectation was that during Year 2 and Year 3 the simulator would be used in running experiments to assess human performance in the use of the novel air traffic control visualisations we would develop. We therefore planned a development of its capabilities to support two types of evaluation activities:

(a) Lab Based Experiments. These types of experiments would have been carried out with non-SME who would have been presented with simplified scenarios. We would have used this to investigate basic properties of the Human Visual Processing system, whose understanding is necessary before moving to the operational environment. The capability to create and modify simple aircraft 3D trajectories will be crucial. Innovative Concepts would be integrated in the simulator to enhance experimental scenario creation.

(b) Realistic simulation. This corresponds to the types of operational usability investigation foreseen for Year 2 and 3, involving SMEs, and based on realistic scenarios. The main requirement for the current simulator is to being able to read the eDEP scenarios. In this way it will be possible to reduce simulation preparation leveraging on the readymade controller validated scenarios devised at EUROCONTROL that will make sure that controllers will be presented with realistic challenges.

Due to the redirection by IRAB, the purpose of this simulator has changed as indicated earlier. With a project focus on developing innovative visualisation and interaction designs to support SESAR both at the controller and aircraft pilot positions. As such, the ATC Sim that has been developed so far is mostly sufficient for our needs. We are unlikely to use the simulator for lab based experiments, and the simulations do not need to be eDEP compatible, or even particularly realistic. We still however, intend to make the movement of the planes more realistic, e.g. turning along a curve rather than about a point, but this realism does not need to be truly veridical.
3. PROBLEMS AND OPPORTUNITIES FOR IMPROVEMENT

Following regular use of the simulator by developers and investigators, the following key problems were highlighted:

(a) The simulator server would only connect to a single client application. This meant that when investigators had finished with one client and wished to start a different one, they were forced to restart the simulator. This was tedious, and wasted valuable time especially when conducting evaluation with users or during demonstrations with users.

(b) The user interface for the simulator did not allow the use of keyboard shortcuts, which, combined with the need for regular restarts, was a significant impediment to usability. Without keyboard shortcuts, this involved using a mouse or a touchpad to navigate through the menu header, selecting sub-menus and then selecting a menu item. While this is a small problem, for the two tasks performed most frequently – loading a scenario and starting the simulator – this became an issue when trying to demonstrate the prototypes to users within a short and limited time frame.

(c) As it was implemented last year, the simulator uses OpenGL to render the evolution of the scenario. The original intention was to run the simulator on one computer and the prototypes on another computer linked over a network. This would facilitate use of any pseudo-pilot interface. It was also very useful during the development and debugging phases. However, when using the simulator and the prototypes on the same computer this visualisation is unnecessary because the simulator is running invisibly in the background. The OpenGL functions incur a high performance overhead, increasing the CPU load, thus making the prototypes perform jerkily on many occasions. Providing an alternate version of the simulator that did not use OpenGL would lower the load on the CPU and smooth the performance of the demonstrations. This would come at the cost of not being able to see what was happening in the simulator.

(d) Adding fixes to the airspace editor was a tedious process that involved entering latitude and longitude into a dialog box. A better solution would be to allow users to position fixes using a point and click interface, that did not require estimations of latitude and longitude.

(e) In the editor, the tree view of the scenario – a listing of aircraft, flight-paths, fixes and clouds – did not update properly. Instead, the tree was static, only updating when the editor was reloaded.
The path following algorithm within the simulator was inadequate. When an aircraft reached a waypoint it would instantly and abruptly turn to face the new direction, instead of turning along a curve as an aircraft would in the real world. The simulated aircraft would follow a series of waypoints, but this following was not smooth. This was adequate for large scale views of the entire airspace, but for concepts that presented close views of few aircraft (e.g. the Holding Stack and Approach Control concepts) this was not acceptable.

In order to complete the path following algorithm it was necessary to change the way that position information was stored within the client. Although the inputs remain as \textit{latitude} (degrees), \textit{longitude} (degrees) and \textit{altitude} (feet), all position information is now stored as kilometres from the origin. This method does not account for curvature of the Earth, so any large scale simulations will be slightly inaccurate, however this was deemed not to be significant for any of our evaluations. Although this change will not be visible to users, it is a significant improvement in the maintainability of the code, and was the basis.

As a result of the change to the data storage, it was decided that the Pseudo-Pilot, an original concept that was not completed then due to time constraints, was now feasible. The pseudo-pilot would allow investigators to change the trajectories of aircraft during a simulation. This would allow investigations in which controllers can actually interact with concepts, rather than passively observing.
4. IMPLEMENTED SOLUTIONS

In this section we briefly describe the solutions we have implemented to the problems identified in the previous section.

4.1 Timeouts

The server needed to be restarted each time we needed to connect a new client to the simulator because previous clients were not properly terminating the TCP/IP connection. There were multiple possible solutions to this problem, including re-writing the client programs, adding support for multiple connections and adding a reset command to the server. It was decided that the simplest solution to the problem was to implement a timeout. If the server does not receive any requests for a given period the connection resets automatically and listens for a new connection. The default time is 5000 milliseconds. This solution has the following critical weakness: If a client makes requests less frequently than the timeout interval, the connection to the server will be lost.

It was especially noted that this could become a problem during start up, where there might be a long delay between making the connection and the first request for data. Five seconds was easily long enough for use with any of the current concepts, however, if this becomes a problem for any future application, the timeout interval can be changed in the settings.properties (see Appendix 1) file in the root directory of the simulator.

4.2 Keyboard Shortcuts

Two key actions – loading a scenario and starting the simulator – happen with such frequency that it was useful to implement keyboard shortcuts. Ctrl-o now opens the Open Scenario dialog, and Ctrl-s now starts the simulation. Ctrl-r restarts the simulation. This saves time during two regularly repeated operations. Previously it required accurate use of a trackpad or mouse. Shortcut keys increase speed and reduce errors.

4.3 Non Graphical Simulation

Since the graphical version of the simulation is normally not necessary during demonstrations and evaluations, it was possible to simply create a new version of the simulator in which all OpenGL content was removed. This had the unforeseen benefit of making this version much easier to install on new machines, since the JOGL (Java OpenGL)
libraries were not necessary. This solution has reduced the CPU usage of the simulator by 50% in tests.

4.4 Scenario Editor Improvements

A new function has been implemented so that users can now place waypoints by clicking on a point in the airspace, instead of entering the waypoint location via a text field in a pop-up window.

The tree is now also refreshed properly after each change to the editor.

4.5 Smooth Path Following

We have implemented a new version of the Seek steering behaviour (Reynolds, 1987) to overcome the problem that the initial path following algorithm did not allow for smooth path following resulting in aircraft making abrupt turns at waypoints.

In this new algorithm, moving objects follow paths by travelling at a maximum speed to the next waypoint in their list. If they have reached that waypoint they set their target as the next waypoint and begin to navigate there. They do this by maintaining their current speed but applying a force to alter their heading and rate of climb/descent. The maximum force can be set in the settings.properties (see Appendix 1) file found in the root directory of the application.

To support this, it was also necessary to change the representation of position and velocity within the simulator to a vector based system using a common metric. In this case, we use kilometre as the common metric. The GeoConverter class was changed to act as a group of static constants for converting between different units. It is important to note that the curvature of the earth is no longer taken into account. This was not considered a significant issue for the type of simulations we will be running during this project.

The changes to internal data storage have not altered the external data inputs – the changes are handled by the programming interfaces and are completely compatible with all existing modules of the simulator.

4.6 The Pseudo-Pilot

This was part of the original specification, but was not completed due to time constraints. With the simplification of the positioning and updating of aircraft, it became feasible to implement a simple pseudo-pilot function. The “pseudo-pilot” module allows an investigator to “fly an aircraft” by changing the aircraft behaviour during a simulation. This means that
participants in the trials can potentially interact with the concepts and directing traffic, rather than simply observing a scenario as it plays out in front of them.

The current implementation for the pseudo-pilot function is a command line interface that allows an investigator to instruct an aircraft to follow a given heading at a specified altitude and velocity. This could easily be improved through the use of a graphic user interface, but that is currently unnecessary, since use is expected to be intermittent (e.g. mainly used during user evaluations of the prototypes) and that only the trained investigators are likely to use this interface to control the aircraft flight in response to instructions from a controller participating in a study.

5. CONCLUSION

The SESAR redirection from IRAB has radically altered the originally intended methods of evaluation, and paradoxically, this means that the usage of the simulator will unlikely be used for experimental investigations, requiring measurement of user responses and responses time. Following the improvements made after the Year 1 ATC Sim and Experimental Testbed was delivered, we consider the ATC Sim to be adequate for use as an ATC traffic scenario generator with the capability for the participating controller in a study to direct traffic through the pseudo-pilot interface. The work for this effort will now be closed.

6. REFERENCES


APPENDIX 1

THE SETTINGS.PROPERTIES FILE

The settings.properties file has the following format:

#The maximum force an aircraft can exert
max_force 5

#The time in milliseconds without a request before a connection can
be reset
timeout 5000

Lines beginning with a "#" are comments.

To change a value, change the number that follows the parameter. Do not change the name of the parameter. If the file is damaged or irreparably changed, a backup can be found in defaults.properties in the same directory. Simply copy all of the text in defaults.properties over the text in settings.properties.

Manipulation of properties is usually accomplished through the use of dialog boxes, in order to simplify the user experience. In this case, the computer literacy of the current end users meant that this would involve unnecessary changes to the program.
APPENDIX 2
ATC SIM SOFTWARE DOCUMENTATION

Below is an example of a java software documentation of a file used in the ATC Sim. This is the SimEngine.java file. In the entire ATC Sim application there are about 80 separate program files. It is not practical to print the entire set of documentation here. They are included with the application in the accompanying CD.
public class SimEngine
extends java.lang.Object

Simulator engine class.

Constructor Summary

SimEngine(AircraftConflictFinder cf, AircraftCloudIntersectionFinder acif)
Constructs a simulation engine, associated to the specified AircraftConflictFinder and AircraftCloudIntersectionFinder instances.

Method Summary

void disableConflictCheck()
Disables the conflict detection.

void disableEventLog()
Disables the event logging.

void doStep(double dt_millis)
Performs the simulation step, using the specified time step in milliseconds.

void enableConflictCheck()
Enables the conflict detection.

void enableEventLog()
Enables the event logging.

java.util.Vector getAircraftCloudIntersections()
Returns the aircraft-cloud intersections detected.

java.util.Vector getConflicts()
Returns the list of aircraft-aircraft conflicts detected.

ClockTime getCurrentTime()
Returns the current simulation time.

int getNumAirplanes()
Returns the number of simulated airplanes.

SimAircraft getSimAirplane(int i)
Returns the simulated aircraft corresponding to the specified index.
java.util.Vector getSimAirplanes()
    Returns the list of simulated airplanes.

ClockTime getSimTime()
    Returns the current simulation time.

ClockTime getStartTime()
    Returns the start time.

void init(Airspace asp)
    Initializes the simulation engine, according to the new specified airspace.

boolean isOver()
    Returns true if the simulation is over; this happens when no active airplanes are available.

void pause()
    Pauses the simulation.

void play()
    Puts the simulation in play mode, disabling the pause mode.

void rewind()
    Rewinds the simulation and pauses it.

void setSimAirplanes(java.util.Vector ap)
    Sets the list of simulated airplanes.

void setStartTime(ClockTime time)
    Sets the start time.

void setTimeSpeed(double s)
    Sets the simulation time speed (1 = real time).

void start()
    Starts (restarts) the simulation.

void startTimer(long dt_millisec)
    Starts the timer, scheduling the periodic execution of the simulation step.

void stop()
    Stops and rewind the simulation.

boolean usingTimer()
    Returns true if this SimEngine instance is using the timer.

Methods inherited from class java.lang.Object
    clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait

Constructor Detail

SimEngine

public SimEngine(AircraftConflictFinder cf,
                 AircraftCloudIntersectionFinder acif)
Constructs a simulation engine, associated to the specified AircraftConflictFinder and AircraftCloudIntersectionFinder instances.

**Method Detail**

**startTimer**

```java
public void startTimer(long dt_millisec)
```

Starts the timer, scheduling the periodic execution of the simulation step.

**usingTimer**

```java
public boolean usingTimer()
```

Returns true if this SimEngine instance is using the timer.

**isOver**

```java
public boolean isOver()
```

Returns true if the simulation is over; this happens when no active airplanes are available.

**setTimeSpeed**

```java
public void setTimeSpeed(double s)
```

Sets the simulation time speed (1 = real time).

**setSimAirplanes**

```java
public void setSimAirplanes(java.util.Vector ap)
```

Sets the list of simulated airplanes.

**setStartTime**

```java
public void setStartTime(ClockTime time)
```

Sets the start time.

**getStartTime**

```java
public ClockTime getStartTime()
```
Returns the start time.

---

**get CURRENTTime**

public ClockTime getCurrentTime()

Returns the current simulation time.

---

**enableConflictCheck**

public void enableConflictCheck()

Enables the conflict detection.

---

**disableConflictCheck**

public void disableConflictCheck()

Disables the conflict detection.

---

**enableEventLog**

public void enableEventLog()

Enables the event logging.

---

**disableEventLog**

public void disableEventLog()

Disables the event logging.

---

**init**

public void init(Airspace asp)

Initializes the simulation engine, according to the new specified airspace. Starts time is automatically computed from the flight plans in the airspace, and current time is set to start time.

---

**getConflicts**

public java.util.Vector getConflicts()

Returns the list of aircraft-aircraft conflicts detected.
**getAircraftCloudIntersections**

```java
public java.util.Vector getAircraftCloudIntersections()
```

Returns the aircraft-cloud intersections detected.

**start**

```java
public void start()
```

Starts (restarts) the simulation.

**stop**

```java
public void stop()
```

Stops and rewind the simulation.

**rewind**

```java
public void rewind()
```

Rewinds the simulation and pauses it.

**pause**

```java
public void pause()
```

Pauses the simulation.

**play**

```java
public void play()
```

Puts the simulation in play mode, disabling the pause mode.

**getSimTime**

```java
public ClockTime getSimTime()
```

Returns the current simulation time.

**getSimAirplanes**
public java.util.Vector getSimAirplanes()

    Returns the list of simulated airplanes.

getSimAirplane

public SimAircraft getSimAirplane(int i)

    Returns the simulated aircraft corresponding to the specified index.

getNumAirplanes

public int getNumAirplanes()

    Returns the number of simulated airplanes.

doStep

public void doStep(double dt_millis)

    Performs the simulation step, using the specified time step in milliseconds.