AirCraft Program Report

Matt Luckcuck

Department of Computer Science,
University of York, UK

ml881@york.ac.uk

4th February 2016

Contents

1 Introduction **2**

2 Program Script **3**
2.1 ACSafelet . . . 3
2.2 MainMissionSequencer . . . 4
2.3 MainMission . . . 5
2.4 Schedulables of MainMission . . . 7
2.4.1 ACModeChanger . . . 7
2.4.2 EnvironmentMonitor . . . 8
2.4.3 ControlHandler . . . 9
2.4.4 FlightSensorsMonitor . . . 10
2.4.5 CommunicationsHandler . . . 11
2.5 TakeOffMission . . . 12
2.6 Schedulables of TakeOffMission . . . 14
2.6.1 TakeOffMonitor . . . 14
2.6.2 TakeOffFailureHandler . . . 15
2.6.3 LandingGearHandlerTakeOff . . . 16
2.7 CruiseMission . . . 17
2.8 Schedulables of CruiseMission . . . 19
2.8.1 NavigationMonitor . . . 19
2.8.2 BeginLandingHandler . . . 20
2.9 LandMission . . . 21
2.10 Schedulables of LandMission . . . 23
2.10.1 GroundDistanceMonitor . . . 23
2.10.2 LandingGearHandlerLand . . . 24
2.10.3 InstrumentLandingSystemMonitor . . . 25
2.10.4 SafeLandingHandler . . . 26

1 Introduction

The *AirCraft* application represents the control software of a simplified aircraft control system, which operates in three modes: take off, cruising, or landing. AirCraft is structurally complicated; making use of Level 2's unique (in SCJ) ability to nest mission sequencers inside missions, which is used to handle the modes of operation. AirCraft is an extended version of an example found in [1].

Figure 1 shows an object diagram of the AirCraft. The program is controlled by the safelet `ACSafelet`, which starts the top-level mission sequencer `MainMissionSequencer`. The mission sequencer starts the `MainMission`, which starts four schedulables and a mission sequencer (`ACModeChanger`). The four schedulables will operate throughout all modes of operation. The `ACModeChanger` starts each of the missions that encapsulate the modes of operation: `TakeOffMission`, `CruiseMission`, and `LandMission`. Each of these missions starts its own schedulables that operate throughout that mode only.

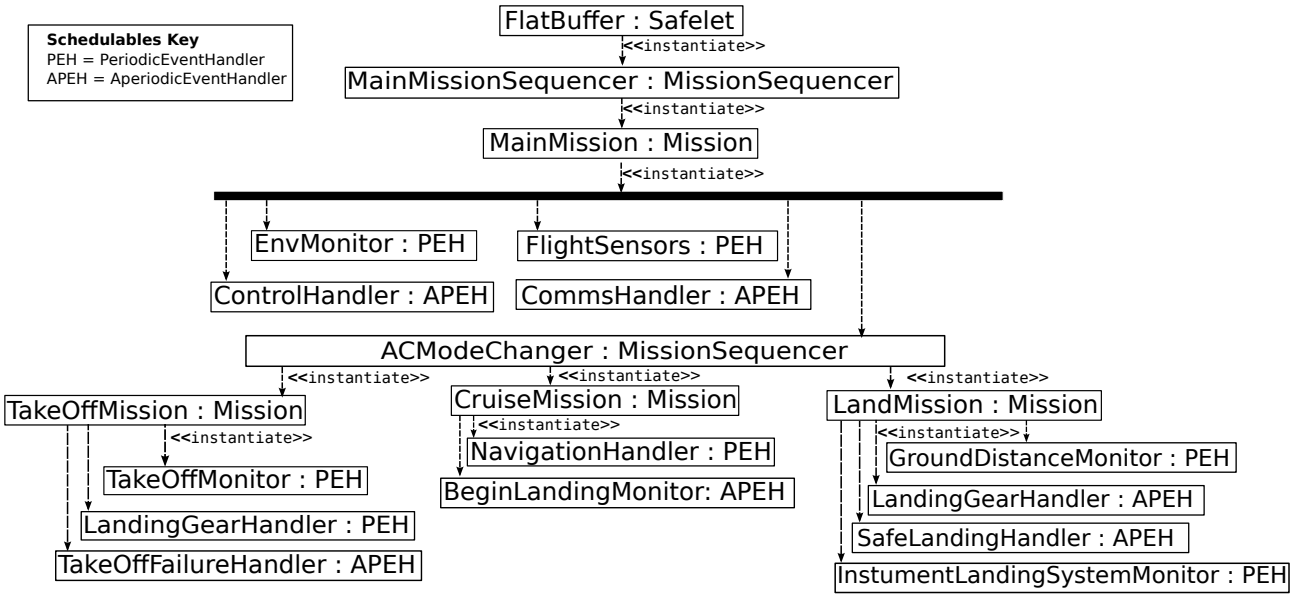

Figure 1: Object Diagram of the AirCraft application

When the AirCraft application begins, the schedulables in the `MainMission` begin executing – including the `ACModeChanger`, which controls three missions that represent the aircraft's modes of operation. Each of the missions controlled by the `ACModeChanger` is requested to terminate, by one of its schedulables, when it's mode of operation is over. Initially the `ACModeChanger` loads the `TakeOffMission`. When it terminates, the `ACModeChanger` loads the next mission. Once the `LandMission` terminates the aircraft has landed, so the program terminates.

The AirCraft application uses the unique feature of Level 2, nested mission sequencers. This feature produces the most obvious hierarchical difference from Level 1 programs. Modes of operation can be captured at Level 1, but Level 2 allows schedulables that run throughout all the modes without interruption. Section 2 presents the full script of the AirCraft program.

2 Program Script

2.1 ACSafelet

```
1  public class ACSafelet implements Safelet<Mission>{
2
3    public MissionSequencer<Mission> getSequencer(){
4      StorageParameters storageParameters = new StorageParameters(150 * 1000,
5          new long[] { Const.HANDLER_STACK_SIZE },
6          Const.PRIVATE_MEM_DEFAULT - 25 * 1000,
7          Const.IMMORTAL_MEM_DEFAULT - 50 * 1000,
8          Const.MISSION_MEM_DEFAULT - 100 * 1000);
9
10     return new MainMissionSequencer(new PriorityParameters(5),
11         storageParameters);
12   }
13
14   public long immortalMemorySize(){
15     return Const.IMMORTAL_MEM_DEFAULT;
16   }
17
18   public void initializeApplication(){
19   }
20 }
```

2.2 MainMissionSequencer

```
1  public class MainMissionSequencer extends MissionSequencer<Mission>{
2
3    private boolean returnedMission;
4
5    public MainMissionSequencer(PriorityParameters priority,
6        StorageParameters storage){
7      super(priority, storage);
8      returnedMission = false;
9    }
10
11   protected Mission getNextMission(){
12
13     if (!returnedMission){
14       returnedMission = true;
15       return new MainMission();
16     } else{
17       return null;
18     }
19   }
20 }
```

2.3 MainMission

```
1  public class MainMission extends Mission{
2    /**
3     * The read that the sensors will get when the aircraft is on the ground
4     */
5    final double ALTITUDE_READING_ON_GROUND = 0.0;
6
7    private double cabinPressure;
8    private double emergencyOxygen;
9    private double fuelRemaining;
10
11   private double altitude;
12   private double airSpeed;
13   private double heading;
14
15   public double getAirSpeed(){
16     return airSpeed;
17   }
18
19   public double getAltitude(){
20     return altitude;
21   }
22
23   public double getCabinPressure(){
24     return cabinPressure;
25   }
26
27   public double getEmergencyOxygen(){
28     return emergencyOxygen;
29   }
30
31   public double getFuelRemaining(){
32     return fuelRemaining;
33   }
34
35   public double getHeading(){
36     return heading;
37   }
38
39   protected void initialize(){
40     StorageParameters storageParameters = new StorageParameters(150 * 1000,
41         new long[] { Const.HANDLER_STACK_SIZE },
42         Const.PRIVATE_MEM_DEFAULT - 25 * 1000,
43         Const.IMMORTAL_MEM_DEFAULT - 50 * 1000,
44         Const.MISSION_MEM_DEFAULT - 100 * 1000);
45
46     StorageParameters storageParametersSchedulable = new StorageParameters(
47         Const.PRIVATE_MEM_DEFAULT - 30 * 1000,
48         new long[] { Const.HANDLER_STACK_SIZE },
49         Const.PRIVATE_MEM_DEFAULT - 30 * 1000,
50         Const.IMMORTAL_MEM_DEFAULT - 50 * 1000,
51         Const.MISSION_MEM_DEFAULT - 100 * 1000);
52
53     ACModeChanger aCModeChanger = new ACModeChanger(new PriorityParameters(
54         5), storageParameters, this);
55
56     aCModeChanger.register();
57
58     EnvironmentMonitor environmentMonitor = new EnvironmentMonitor(
59         new PriorityParameters(5), new PeriodicParameters(
60             new RelativeTime(10, 0), null),
61         storageParametersSchedulable, "Environment Monitor", this);
62
63     environmentMonitor.register();
64
65     ControlHandler controlHandler = new ControlHandler(
66         new PriorityParameters(5), new AperiodicParameters(new RelativeTime(10, 0), null),
67         storageParametersSchedulable, "Control Handler");
68
69     controlHandler.register();
70
71     FlightSensorsMonitor flightSensMon = new FlightSensorsMonitor(
72         new PriorityParameters(5), new PeriodicParameters(
```

```
73                  new RelativeTime(10, 0), null),
74              storageParametersSchedulable, "Flight Sensors Monitor", this);
75
76      flightSensMon.register();
77
78      CommunicationsHandler commsHandler = new CommunicationsHandler(
79          new PriorityParameters(5), new AperiodicParameters(),
80          storageParametersSchedulable, "Communications Handler");
81
82      commsHandler.register();
83
84      AperiodicSimulator controlSim
85        = new AperiodicSimulator(
86          new PriorityParameters(5),
87          new PeriodicParameters(new RelativeTime(10, 0), null),
88          storageParametersSchedulable,
89          controlHandler);
90
91      controlSim.register();
92
93    }
94
95    public long missionMemorySize(){
96      return Const.MISSION_MEM_DEFAULT;
97    }
98
99    public void setAirSpeed(double airSpeed){
100     this.airSpeed = airSpeed;
101   }
102
103   public void setAltitude(double altitude){
104     this.altitude = altitude;
105   }
106
107   public void setCabinPressure(double cabinPressure){
108     this.cabinPressure = cabinPressure;
109   }
110
111   public void setEmergencyOxygen(double emergencyOxygen){
112     this.emergencyOxygen = emergencyOxygen;
113   }
114
115   public void setFuelRemaining(double fuelRemaining){
116     this.fuelRemaining = fuelRemaining;
117   }
118
119   public void setHeading(double heading){
120     this.heading = heading;
121   }
122 }
```

2.4 Schedulables of MainMission

2.4.1 ACModeChanger

```
1  public class ACModeChanger extends MissionSequencer<Mission>{
2
3    private MainMission controllingMission;
4    private int modesLeft = 3;
5
6    public ACModeChanger(PriorityParameters priority,
7        StorageParameters storage, MainMission controllingMission){
8      super(priority, storage);
9      this.controllingMission = controllingMission;
10   }
11
12   public ACModeChanger(PriorityParameters priority, StorageParameters storage){
13     super(priority, storage);
14   }
15
16   protected Mission getNextMission(){
17     if (modesLeft == 3)    {
18       modesLeft--;
19       return new TakeOffMission(controllingMission);
20     } else if (modesLeft == 2){
21       modesLeft--;
22       return new CruiseMission(controllingMission);
23     } else if (modesLeft == 1){
24       modesLeft--;
25       return new LandMission(controllingMission);
26     } else{
27       return null;
28     }
29   }
30 }
```

2.4.2 EnvironmentMonitor

```
1  public class EnvironmentMonitor extends PeriodicEventHandler{
2
3    MainMission controllingMission;
4
5    public EnvironmentMonitor(PriorityParameters priority,
6        PeriodicParameters periodic,
7        StorageParameters storage,
8        String name,
9        MainMission mainMission){
10     super(priority, periodic, storage);
11     controllingMission = mainMission;
12   }
13
14   public void handleAsyncEvent(){
15     System.out.println("Checking Environment");
16
17     // read cabin pressure from sensors
18     controllingMission.setCabinPressure(0);
19
20     // read emergency Oxygen Levels
21     controllingMission.setEmergencyOxygen(0);
22
23     // read remaining fuel
24     controllingMission.setFuelRemaining(0);
25   }
26 }
```

2.4.3 ControlHandler

```
1  public class ControlHandler extends AperiodicEventHandler{
2
3    public ControlHandler(PriorityParameters priority,
4        AperiodicParameters release, StorageParameters storage, String name){
5      super(priority, release, storage, name);
6    }
7
8    public void handleAsyncEvent(){
9      System.out.println("Handling Controls");
10   }
11 }
```

2.4.4 FlightSensorsMonitor

```
1  public class FlightSensorsMonitor extends PeriodicEventHandler{
2
3    MainMission controllingMission;
4
5    public FlightSensorsMonitor(PriorityParameters priority,
6        PeriodicParameters periodic, StorageParameters storage,
7        String name, MainMission mainMission){
8      super(priority, periodic, storage);
9      controllingMission = mainMission;
10   }
11
12   public void handleAsyncEvent(){
13
14     System.out.println("Checking Flight Sensors");
15
16     // read air speed
17     controllingMission.setAirSpeed(0);
18     // read altitude
19     controllingMission.setAltitude(0);
20     // read heading
21     controllingMission.setHeading(0);
22   }
23 }
```

2.4.5 CommunicationsHandler

```
1  public class CommunicationsHandler extends AperiodicEventHandler{
2
3    public CommunicationsHandler(PriorityParameters priority,
4        AperiodicParameters release, StorageParameters storage, String name){
5      super(priority, release, storage, name);
6    }
7
8    public void handleAsyncEvent(){
9      System.out.println("Handling Comms");
10   }
11 }
```

2.5 TakeOffMission

```
1  public class MainMission extends Mission{
2    /**
3     * The read that the sensors will get when the aircraft is on the ground
4     */
5    final double ALTITUDE_READING_ON_GROUND = 0.0;
6
7    private double cabinPressure;
8    private double emergencyOxygen;
9    private double fuelRemaining;
10
11   private double altitude;
12   private double airSpeed;
13   private double heading;
14
15   public double getAirSpeed(){
16     return airSpeed;
17   }
18
19   public double getAltitude(){
20     return altitude;
21   }
22
23   public double getCabinPressure(){
24     return cabinPressure;
25   }
26
27   public double getEmergencyOxygen(){
28     return emergencyOxygen;
29   }
30
31   public double getFuelRemaining(){
32     return fuelRemaining;
33   }
34
35   public double getHeading(){
36     return heading;
37   }
38
39   protected void initialize(){
40     StorageParameters storageParameters = new StorageParameters(150 * 1000,
41         new long[] { Const.HANDLER_STACK_SIZE },
42         Const.PRIVATE_MEM_DEFAULT - 25 * 1000,
43         Const.IMMORTAL_MEM_DEFAULT - 50 * 1000,
44         Const.MISSION_MEM_DEFAULT - 100 * 1000);
45
46     StorageParameters storageParametersSchedulable = new StorageParameters(
47         Const.PRIVATE_MEM_DEFAULT - 30 * 1000,
48         new long[] { Const.HANDLER_STACK_SIZE },
49         Const.PRIVATE_MEM_DEFAULT - 30 * 1000,
50         Const.IMMORTAL_MEM_DEFAULT - 50 * 1000,
51         Const.MISSION_MEM_DEFAULT - 100 * 1000);
52
53     ACModeChanger aCModeChanger = new ACModeChanger(new PriorityParameters(
54         5), storageParameters, this);
55
56     aCModeChanger.register();
57
58     EnvironmentMonitor environmentMonitor = new EnvironmentMonitor(
59         new PriorityParameters(5), new PeriodicParameters(
60             new RelativeTime(10, 0), null),
61         storageParametersSchedulable, "Environment Monitor", this);
62
63     environmentMonitor.register();
64
65     ControlHandler controlHandler = new ControlHandler(
66         new PriorityParameters(5), new AperiodicParameters(new RelativeTime(10, 0), null),
67         storageParametersSchedulable, "Control Handler");
68
69     controlHandler.register();
70
71     FlightSensorsMonitor flightSensMon = new FlightSensorsMonitor(
72         new PriorityParameters(5), new PeriodicParameters(
```

```
73                 new RelativeTime(10, 0), null),
74             storageParametersSchedulable, "Flight Sensors Monitor", this);
75
76       flightSensMon.register();
77
78       CommunicationsHandler commsHandler = new CommunicationsHandler(
79           new PriorityParameters(5), new AperiodicParameters(),
80           storageParametersSchedulable, "Communications Handler");
81
82       commsHandler.register();
83
84       AperiodicSimulator controlSim
85         = new AperiodicSimulator(
86           new PriorityParameters(5),
87           new PeriodicParameters(new RelativeTime(10, 0), null),
88           storageParametersSchedulable,
89           controlHandler);
90
91       controlSim.register();
92
93     }
94
95     public long missionMemorySize(){
96       return Const.MISSION_MEM_DEFAULT;
97     }
98
99     public void setAirSpeed(double airSpeed){
100      this.airSpeed = airSpeed;
101    }
102
103    public void setAltitude(double altitude){
104      this.altitude = altitude;
105    }
106
107    public void setCabinPressure(double cabinPressure){
108      this.cabinPressure = cabinPressure;
109    }
110
111    public void setEmergencyOxygen(double emergencyOxygen){
112      this.emergencyOxygen = emergencyOxygen;
113    }
114
115    public void setFuelRemaining(double fuelRemaining){
116      this.fuelRemaining = fuelRemaining;
117    }
118
119    public void setHeading(double heading){
120      this.heading = heading;
121    }
122  }
```

2.6 Schedulables of TakeOffMission

2.6.1 TakeOffMonitor

```
1  public class TakeOffMonitor extends PeriodicEventHandler{
2
3    private final MainMission mainMission ;
4    private final TakeOffMission takeoffMission;
5
6    private double takeOffAltitude;
7    private AperiodicEventHandler landingGearHandler;
8
9    public TakeOffMonitor(PriorityParameters priority,
10       PeriodicParameters periodic, StorageParameters storage,
11       MainMission mainMission, TakeOffMission takeOffMission, double takeOffAltitude,
12       AperiodicEventHandler landingGearHandler){
13     super(priority, periodic, storage);
14     this.mainMission = mainMission;
15     this.takeoffMission = takeOffMission;
16     this.takeOffAltitude = takeOffAltitude;
17     this.landingGearHandler = landingGearHandler;
18   }
19
20   public void handleAsyncEvent(){
21     System.out.println("Reading Altitude");
22     double altitude = mainMission.getAltitude();
23
24     if (altitude > takeOffAltitude){
25       System.out.println("Take Off Complete");
26       landingGearHandler.release();
27       takeoffMission.requestTermination();
28     }
29   }
30 }
```

2.6.2 TakeOffFailureHandler

```
1  public class TakeOffFailureHandler extends AperiodicEventHandler{
2
3    private final MainMission mainMission;
4    private final TakeOffMission takeoffMission;
5    private double threshold;
6
7    public TakeOffFailureHandler(PriorityParameters priority,
8        AperiodicParameters release, StorageParameters storage,
9        String name, MainMission mainMission, TakeOffMission takeoffMission, Double threshold){
10     super(priority, release, storage, name);
11     this.takeoffMission = takeoffMission;
12     this.mainMission = mainMission;
13     this.threshold = threshold;
14   }
15
16   public void handleAsyncEvent(){
17
18     double currentSpeed = mainMission.getAirSpeed();
19
20     // in both cases this failure should be flagged somewhere
21     if (currentSpeed < threshold){
22       System.out.println("Failure: Aborting");
23       takeoffMission.abort();
24       takeoffMission.requestTermination();
25     } else{
26       System.out.println("Failure: Continue and Land");
27     }
28   }
29 }
```

2.6.3 LandingGearHandlerTakeOff

```
1  public class LandingGearHandlerTakeOff extends AperiodicEventHandler{
2
3    private final TakeOffMission mission;
4
5    public LandingGearHandlerTakeOff(PriorityParameters priority,
6        AperiodicParameters release, StorageParameters storage,
7        String name, TakeOffMission mission){
8      super(priority, release, storage, name);
9      this.mission = mission;
10   }
11
12   public void handleAsyncEvent(){
13
14     System.out.println("Deploying Landing Gear");
15
16     boolean landingGearIsDeployed = mission.isLandingGearDeployed();
17
18     if (landingGearIsDeployed){
19       mission.stowLandingGear();
20     } else{
21       mission.deployLandingGear();
22     }
23   }
24 }
```

2.7 CruiseMission

```
1  public class MainMission extends Mission{
2    /**
3     * The read that the sensors will get when the aircraft is on the ground
4     */
5    final double ALTITUDE_READING_ON_GROUND = 0.0;
6
7    private double cabinPressure;
8    private double emergencyOxygen;
9    private double fuelRemaining;
10
11   private double altitude;
12   private double airSpeed;
13   private double heading;
14
15   public double getAirSpeed(){
16     return airSpeed;
17   }
18
19   public double getAltitude(){
20     return altitude;
21   }
22
23   public double getCabinPressure(){
24     return cabinPressure;
25   }
26
27   public double getEmergencyOxygen(){
28     return emergencyOxygen;
29   }
30
31   public double getFuelRemaining(){
32     return fuelRemaining;
33   }
34
35   public double getHeading(){
36     return heading;
37   }
38
39   protected void initialize(){
40     StorageParameters storageParameters = new StorageParameters(150 * 1000,
41         new long[] { Const.HANDLER_STACK_SIZE },
42         Const.PRIVATE_MEM_DEFAULT - 25 * 1000,
43         Const.IMMORTAL_MEM_DEFAULT - 50 * 1000,
44         Const.MISSION_MEM_DEFAULT - 100 * 1000);
45
46     StorageParameters storageParametersSchedulable = new StorageParameters(
47         Const.PRIVATE_MEM_DEFAULT - 30 * 1000,
48         new long[] { Const.HANDLER_STACK_SIZE },
49         Const.PRIVATE_MEM_DEFAULT - 30 * 1000,
50         Const.IMMORTAL_MEM_DEFAULT - 50 * 1000,
51         Const.MISSION_MEM_DEFAULT - 100 * 1000);
52
53     ACModeChanger aCModeChanger = new ACModeChanger(new PriorityParameters(
54         5), storageParameters, this);
55
56     aCModeChanger.register();
57
58     EnvironmentMonitor environmentMonitor = new EnvironmentMonitor(
59         new PriorityParameters(5), new PeriodicParameters(
60             new RelativeTime(10, 0), null),
61         storageParametersSchedulable, "Environment Monitor", this);
62
63     environmentMonitor.register();
64
65     ControlHandler controlHandler = new ControlHandler(
66         new PriorityParameters(5), new AperiodicParameters(new RelativeTime(10, 0), null),
67         storageParametersSchedulable, "Control Handler");
68
69     controlHandler.register();
70
71     FlightSensorsMonitor flightSensMon = new FlightSensorsMonitor(
72         new PriorityParameters(5), new PeriodicParameters(
```

```
73                  new RelativeTime(10, 0), null),
74              storageParametersSchedulable, "Flight Sensors Monitor", this);
75
76      flightSensMon.register();
77
78      CommunicationsHandler commsHandler = new CommunicationsHandler(
79          new PriorityParameters(5), new AperiodicParameters(),
80          storageParametersSchedulable, "Communications Handler");
81
82      commsHandler.register();
83
84      AperiodicSimulator controlSim
85        = new AperiodicSimulator(
86          new PriorityParameters(5),
87          new PeriodicParameters(new RelativeTime(10, 0), null),
88          storageParametersSchedulable,
89          controlHandler);
90
91      controlSim.register();
92
93    }
94
95    public long missionMemorySize(){
96      return Const.MISSION_MEM_DEFAULT;
97    }
98
99    public void setAirSpeed(double airSpeed){
100     this.airSpeed = airSpeed;
101   }
102
103   public void setAltitude(double altitude){
104     this.altitude = altitude;
105   }
106
107   public void setCabinPressure(double cabinPressure){
108     this.cabinPressure = cabinPressure;
109   }
110
111   public void setEmergencyOxygen(double emergencyOxygen){
112     this.emergencyOxygen = emergencyOxygen;
113   }
114
115   public void setFuelRemaining(double fuelRemaining){
116     this.fuelRemaining = fuelRemaining;
117   }
118
119   public void setHeading(double heading){
120     this.heading = heading;
121   }
122 }
```

2.8 Schedulables of CruiseMission

2.8.1 NavigationMonitor

```
1  public class NavigationMonitor extends PeriodicEventHandler{
2
3    private final MainMission mainMission;
4
5    public NavigationMonitor(PriorityParameters priority,
6        PeriodicParameters periodic, StorageParameters storage,
7        String name, MainMission mainMission){
8      super(priority, periodic, storage);
9      this.mainMission = mainMission;
10   }
11
12   public void handleAsyncEvent(){
13     // read and check these variables
14     double heading = mainMission.getHeading();
15     double airSpeed = mainMission.getAirSpeed();
16     double altitude = mainMission.getAltitude();
17
18     // Obviously this would then check the variables again expected values
19   }
20 }
```

2.8.2 BeginLandingHandler

```
1  public class BeginLandingHandler extends AperiodicEventHandler{
2
3    private Mission controllingMission;
4
5    public BeginLandingHandler(PriorityParameters priority,
6        AperiodicParameters release, StorageParameters storage,
7        String name, Mission controllingMission){
8      super(priority, release, storage, name);
9      this.controllingMission = controllingMission;
10   }
11
12   public void handleAsyncEvent(){
13     System.out.println("Begin Landing");
14     controllingMission.requestTermination();
15   }
16 }
```

2.9 LandMission

```
1  public class MainMission extends Mission{
2    /**
3     * The read that the sensors will get when the aircraft is on the ground
4     */
5    final double ALTITUDE_READING_ON_GROUND = 0.0;
6
7    private double cabinPressure;
8    private double emergencyOxygen;
9    private double fuelRemaining;
10
11   private double altitude;
12   private double airSpeed;
13   private double heading;
14
15   public double getAirSpeed(){
16     return airSpeed;
17   }
18
19   public double getAltitude(){
20     return altitude;
21   }
22
23   public double getCabinPressure(){
24     return cabinPressure;
25   }
26
27   public double getEmergencyOxygen(){
28     return emergencyOxygen;
29   }
30
31   public double getFuelRemaining(){
32     return fuelRemaining;
33   }
34
35   public double getHeading(){
36     return heading;
37   }
38
39   protected void initialize(){
40     StorageParameters storageParameters = new StorageParameters(150 * 1000,
41         new long[] { Const.HANDLER_STACK_SIZE },
42         Const.PRIVATE_MEM_DEFAULT - 25 * 1000,
43         Const.IMMORTAL_MEM_DEFAULT - 50 * 1000,
44         Const.MISSION_MEM_DEFAULT - 100 * 1000);
45
46     StorageParameters storageParametersSchedulable = new StorageParameters(
47         Const.PRIVATE_MEM_DEFAULT - 30 * 1000,
48         new long[] { Const.HANDLER_STACK_SIZE },
49         Const.PRIVATE_MEM_DEFAULT - 30 * 1000,
50         Const.IMMORTAL_MEM_DEFAULT - 50 * 1000,
51         Const.MISSION_MEM_DEFAULT - 100 * 1000);
52
53     ACModeChanger aCModeChanger = new ACModeChanger(new PriorityParameters(
54         5), storageParameters, this);
55
56     aCModeChanger.register();
57
58     EnvironmentMonitor environmentMonitor = new EnvironmentMonitor(
59         new PriorityParameters(5), new PeriodicParameters(
60             new RelativeTime(10, 0), null),
61         storageParametersSchedulable, "Environment Monitor", this);
62
63     environmentMonitor.register();
64
65     ControlHandler controlHandler = new ControlHandler(
66         new PriorityParameters(5), new AperiodicParameters(new RelativeTime(10, 0), null),
67         storageParametersSchedulable, "Control Handler");
68
69     controlHandler.register();
70
71     FlightSensorsMonitor flightSensMon = new FlightSensorsMonitor(
72         new PriorityParameters(5), new PeriodicParameters(
```

```
73                  new RelativeTime(10, 0), null),
74              storageParametersSchedulable, "Flight Sensors Monitor", this);
75
76      flightSensMon.register();
77
78      CommunicationsHandler commsHandler = new CommunicationsHandler(
79          new PriorityParameters(5), new AperiodicParameters(),
80          storageParametersSchedulable, "Communications Handler");
81
82      commsHandler.register();
83
84      AperiodicSimulator controlSim
85        = new AperiodicSimulator(
86          new PriorityParameters(5),
87          new PeriodicParameters(new RelativeTime(10, 0), null),
88          storageParametersSchedulable,
89          controlHandler);
90
91      controlSim.register();
92
93    }
94
95    public long missionMemorySize(){
96      return Const.MISSION_MEM_DEFAULT;
97    }
98
99    public void setAirSpeed(double airSpeed){
100     this.airSpeed = airSpeed;
101   }
102
103   public void setAltitude(double altitude){
104     this.altitude = altitude;
105   }
106
107   public void setCabinPressure(double cabinPressure){
108     this.cabinPressure = cabinPressure;
109   }
110
111   public void setEmergencyOxygen(double emergencyOxygen){
112     this.emergencyOxygen = emergencyOxygen;
113   }
114
115   public void setFuelRemaining(double fuelRemaining){
116     this.fuelRemaining = fuelRemaining;
117   }
118
119   public void setHeading(double heading){
120     this.heading = heading;
121   }
122 }
```

2.10 Schedulables of LandMission

2.10.1 GroundDistanceMonitor

```
1  public class GroundDistanceMonitor extends PeriodicEventHandler{
2
3    private final MainMission mainMission;
4    private final double readingOnGround;
5
6    public GroundDistanceMonitor(PriorityParameters priority,
7        PeriodicParameters periodic, StorageParameters storage,
8        MainMission mainMission){
9      super(priority, periodic, storage);
10
11     this.mainMission = mainMission;
12     this.readingOnGround = mainMission.ALTITUDE_READING_ON_GROUND;
13   }
14
15   public void handleAsyncEvent(){
16
17     System.out.println("Checking Ground Distance");
18     // read this value from sensors
19     double distance = mainMission.getAltitude();
20
21     if (distance == readingOnGround){
22
23       System.out.println("Aircraft Landed, Terminating Mission");
24       mainMission.requestTermination();
25     }
26   }
27 }
```

2.10.2 LandingGearHandlerLand

```
1  public class LandingGearHandlerLand extends AperiodicEventHandler{
2
3    private final LandMission mission;
4
5    public LandingGearHandlerLand(PriorityParameters priority,
6        AperiodicParameters release, StorageParameters storage,
7        String name, LandMission mission){
8      super(priority, release, storage, name);
9      this.mission = mission;
10   }
11
12   public void handleAsyncEvent(){
13
14     System.out.println("Deploying Landing Gear");
15
16     boolean landingGearIsDeployed = mission.isLandingGearDeployed();
17
18     if (landingGearIsDeployed){
19       mission.stowLandingGear();
20     } else{
21       mission.deployLandingGear();
22     }
23   }
24 }
```

2.10.3 InstrumentLandingSystemMonitor

```
1  public class InstrumentLandingSystemMonitor extends PeriodicEventHandler{
2
3    private final LandMission mission;
4
5    public InstrumentLandingSystemMonitor(PriorityParameters priority,
6        PeriodicParameters periodic, StorageParameters storage,
7        String name, LandMission mission){
8      super(priority, periodic, storage);
9      this.mission = mission;
10   }
11
12   public void handleAsyncEvent(){
13     System.out.println("Checking ILS");
14   }
15 }
```

2.10.4 SafeLandingHandler

```
1  public class SafeLandingHandler extends AperiodicEventHandler{
2
3    private final MainMission mainMission;
4    private double threshold;
5
6    public SafeLandingHandler(PriorityParameters priority,
7        AperiodicParameters release, StorageParameters storage,
8        String name, MainMission mainMission, Double threshold){
9      super(priority, release, storage, name);
10     this.mainMission = mainMission;
11     this.threshold = threshold;
12   }
13
14   public void handleAsyncEvent(){
15
16     double altitude = mainMission.getAltitude();
17
18     // in both cases this failure should be flagged somewhere
19     if (altitude < threshold){
20       System.out.println("Failure: Pull Up");
21       // Also perform some recovery action here, maybe a new mode
22     } else{
23       System.out.println("Failure: Continue With Landing");
24     }
25   }
26 }
```

References

[1] Andy Wellings, Matt Luckcuck, and Ana Cavalcanti. Safety-critical java level 2: motivations, example applications and issues. In *Proceedings of the 11th International Workshop on Java Technologies for Real-time and Embedded Systems*, JTRES '13, pages 48–57, New York, NY, USA, 9 October 2013. ACM.