2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)

A Model-based Testing Approach for Cockpit Display Systems of Avionics

Muhammad Zohaib Iqbal*†, Hassan Sartaj*†, Muhammad Uzair Khan*†, Fitash Ul Haq‡ and Ifrah Qaisar§
*Quest Lab, National University of Computer and Emerging Sciences
† UAV Dependability Lab, National Center of Robotics and Automation (NCRA)
FAST Foundation, Rohtas Road, G-9/4, Islamabad - 44000, Pakistan
Email: {zohaib.iqbal,hassan.sartaj,uzair.khan}@questlab.pk
‡University of Luxembourg
Luxembourg City, Luxembourg
Email: fitash.ulhaq@uni.lu
§National University of Computer and Emerging Sciences
A.K. Brohi Road, H-11/4, Islamabad - 44000, Pakistan
Email: ifrah.qaisar@nu.edu.pk

***Abstract*—Avionics are highly critical systems that require extensive testing governed by international safety standards. Cockpit Display Systems (CDS) are an essential component of modern aircraft cockpits and display information from the user application (UA) using various widgets. A significant step in the testing of avionics is to evaluate whether these CDS are displaying the correct information. A common industrial practice is to manually test the information on these CDS by taking the aircraft into different scenarios during the simulation. Such testing is required very frequently and at various changes in the avionics. Given the large number of scenarios to test, manual testing of such behavior is a laborious activity. In this paper, we propose a model-based strategy for automated testing of the information displayed on CDS. Our testing approach focuses on evaluating that the information from the user applications is being displayed correctly on the CDS. For this purpose, we develop a profile for capturing the details of different widgets of the display screens using models. The profile is based on the ARINC 661 standard for Cockpit Display Systems. The expected behavior of the CDS visible on the screens of the aircraft is captured using constraints written in Object Constraint Language. We apply our approach on an industrial case study of a Primary Flight Display (PFD) developed for an aircraft. Our results showed that the proposed approach is able to automatically identify faults in the simulation of PFD. Based on the results, it is concluded that the proposed approach is useful in finding display faults on avionics CDS.**

***Index Terms*—Model-based Testing; Cockpit Display Systems; Safety-critical Systems; ARINC 661; Object Constraint Language (OCL);**

I. Introduction

Avionics software systems need to meet the quality requirements set by various international safety standards [1]. To meet the safety requirements of the standard, the testing and verification of avionics software require an extensive amount of efforts and costs [2]. A significant enhancement to the modern-day aircrafts is the introduction of a glass cockpit that comprises of a Cockpit Display Systems (CDS). These CDS are a replacement of a number of dials and gauges in the traditional aircrafts [3].

These CDS display information that is vital for the safe operation of an aircraft. This may include information coming from different user applications, the flight management system, flight control unit and the warnings generated by different hardware components. Testing that the information displayed on the CDS is correct is an important part of the overall testing activities of an aircraft. One major challenge in testing CDS is that the information displayed on CDS heavily relies on the flight behavior of an aircraft. Another important challenge is the classification of correct and incorrect information. The information of CDS that is made visible to the pilots may vary significantly from scenario to scenario. For example, during the taxi before takeoff, a Takeoff Memo appears on the screen that shows various instructions for the pilot. After the takeoff, the screen disappears. Similarly, when the aircraft turns into a 45° angle (a steep turns), the bank pointer shows a warning by changing the color to amber.

A common practice by the aircraft vendors is to test the information displayed on CDS by manually executing different aircraft scenarios and manually verifying that correct information is displayed according to these scenarios [4]. The scenarios are typically executed with the help of simulators. This step has to be performed repeatedly whenever the required information to be displayed is changed, for example, due to an upgraded sensor being used. Testing in this way (manual execution and manual verification of results) is a very time consuming and laborious task.

In this paper, we propose a model-based automated approach to test the functionality of CDS by evaluating the information displayed on CDS. Our testing focuses on verifying that the information from the user applications is being displayed correctly on the CDS. For this purpose, we develop a UML profile for the international standard of cockpit displays, the ARINC 661 standard [5], to capture the various elements of a CDS. The CDS under test is modeled using the proposed UML profile. The instance model corresponding to the CDS is automatically populated from the existing CDS modeling

978-1-7281-2536-7/19/$31.00 ©2019 IEEE
DOI 10.1109/MODELS.2019.00-14

tools, such as VAPS XT [6]. The test engineer is required to model the different aircraft flight states that have an impact on the CDS elements by using a state machine. The expected properties of the various CDS elements during the aircraft flight are modeled as constraints, written in Object Constraint Language (OCL) [7]. The approach utilizes the developed state machine to generate the test paths. The OCL constraints contain the expected values and are used as an oracle. The test execution is also automated with the help of flight simulators. The actual values displayed on the widgets of a CDS are identified using our image processing and optical character recognition (OCR) tool. Based on the inputs from the OCR tool during different stages of flight, a number of instance models are populated. We use an OCL Evaluator to evaluate the constraints on each of the instance models. Any instance model that does not satisfy the specified OCL constraints represents a failed test case.

We apply our approach on an industrial case study of a Primary Flight Display (PFD) developed for an aircraft. We use JSBSim [8] for simulating the behavior of actual aircraft during testing. Results indicate that our approach is viable and is able to successfully detect 18 faults in the implementation of PFD.

To summarize, the main contributions of this paper are:

1) We propose a model-based approach for functional testing of the cockpit display system (CDS) of an aircraft.
2) We develop a UML profile to capture the information displayed on CDS. The models developed using the profile are then use for specifying oracle (expected values), guiding the test execution tool, and generating instances during test generation.
3) We develop a tool to automate our approach for testing CDS.
4) We apply the proposed strategy on an industrial case study of a Primary Flight Display (PFD) of an aircraft.

The remaining part of the paper is organized as follows. Section II presents a background of model-based testing (MBT), cockpit display system (CDS), and ARINC 661 standard. Section III describes our proposed model-based testing strategy for cockpit display system (CDS) of avionics. Section IV provides a discussion on tool support. Section V presents the evaluation of the proposed approach. Section VI discusses the limitations of the proposed approach. Section VII provides related work. Finally, Section VIII concludes the paper.

II. Background

In this section, we provide the background for model-based testing (MBT), cockpit display system (CDS), and the ARINC 661 standard.

A. Model-based Testing

Model-based testing (MBT) provides a systematic way to automate testing activities [9], [10]. In MBT, the system specifications are modeled using a modeling language such as Unified Modeling Langauge (UML) [11]. To model various

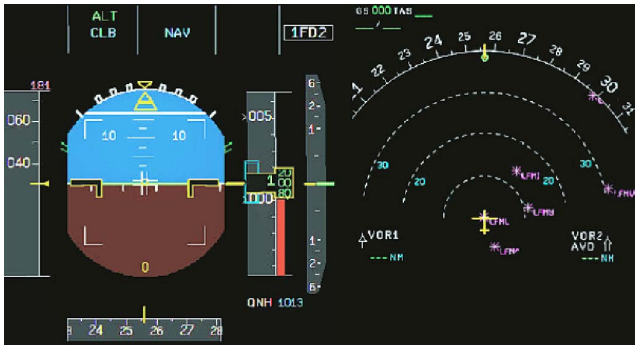

Fig. 1. A primary flight display (PFD) of Airbus A320

aspects of the system, UML provides a number of modeling artifacts for different purposes. The modeling artifacts provided by UML are broadly categorized as structural models (e.g., class diagram and profile diagram) and behavioral models (e.g., state machine). The models developed in UML are augmented using a constraints specification language, i.e., Object Constraint Language (OCL) [7]. Different UML models support the automation of a number of testing activities. For example, the UML state machine can be used to generate test sequences [12]. Similarly, the UML class diagram along with OCL can be used to automatically generate the test data [13].

B. Cockpit Display System (CDS)

The cockpit of an aircraft typically consists of a number of display elements to show various types of information (e.g., altitude) graphically. The display elements consist of primary flight display (PFD), navigation display, altimeter, speed indicator, heading indicator, etc. For example, Fig. 1 shows a primary flight display (PFD) used in Airbus A320[1]. On the left-hand side of the PFD shown in Fig. 1, the information displayed consists of altitude, airspeed, vertical speed, and heading indicator. On the right-hand side of the PFD, the navigation information containing waypoints, direction, and distance is displayed.

C. ARINC 661

ARINC 661 [5] is an aviation standard that defines a method to design the interactive displays for the Cockpit Display System (CDS) of an aircraft. This standard emphasizes separating the user interface (UI) from the application logic. For this purpose, the standard provides a widgets library to design UI of CDS. The standard also defines the protocol to perform communication between UI and the application. The user application receives data from different hardware components (e.g., sensors) and sends data to display system. Moreover, the user application receives commands generated for each interaction of a user on CDS and transfer those commands to the appropriate component.

[1]https://cockpitsonic.de/a320-table-trainer/attachment/a320-pfd

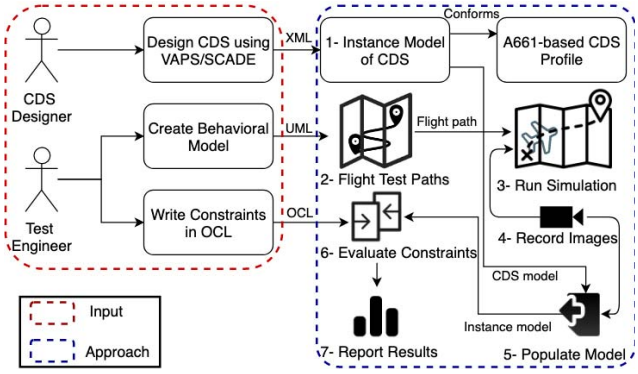

Fig. 2. An overview of the proposed approach

TABLE I
PROFILE STEREOTYPE DESCRIPTION

Stereotype	Description
«Label»	It defines a non-editable text field at a specific location.
«GPDiamond»	It defines a small diamond that is used to display heading information of the Aircraft.
«Altimeter»	It is used to define the concept of Altitude tape that displays the altitude of the aircraft above mean sea level.
«AttitudeIndicator-Display»	Attitude indicator display presents information related to aircraft pitch, roll, and positioning of the aircraft with respect to the horizon.
«AirspeedIndicator»	It represents the indication of the aircraft speed in knots.

III. CDS TESTING APPROACH

In this section, we present the proposed model-based approach for automated testing of the information presented on the Multifunction displays in aircraft cockpits. For automated testing, a multi-step approach is proposed. First, we provide an overview of the complete approach followed by the discussion on each step of the proposed approach.

A. Approach Overview

As shown in Fig. 2, as a first step, the CDS modeled in a graphical modeling tool (such as VAPS XT [6], or SCADE [14]) are converted to an instance of CDS model that conforms our proposed CDS modeling profile. The next step is to model the behavior specification (state machine) of the possible states of an aircraft during the flight that has an impact on the information being displayed on CDS. The third step is to model the constraints on the CDS models. These constraints behave like the test oracle and must be true during some specific states of the aircraft. The state machine is used for the automated generation of test cases (flight test paths). According to each flight path, the behavior of an aircraft flight is simulated. During simulation, the information displayed on CDS is recorded in the form of images. The data from images is extracted to populate the CDS instance models. Lastly, OCL constraints are evaluated on CDS instance models and results are reported. Following we discuss the approach in more detail.

B. A UML Profile for Cockpit Display Systems (CDS)

Our profile is based on the well-established standards for CDS, referred to as the ARINC 661 (or A661) [5]. The A661 standard defines a set of standard widgets and constraints for the development of CDS. The profile defines different concepts and attributes using the A661 Widget Library [15]. The purpose of the profile is to capture the details of the CDS, including the various components that are rendered on the CDS including the widgets and possible values (for example, alignment value, color, position).

Fig. 3 shows an excerpt of the UML profile that we developed for modeling the cockpit display system. The core stereotype of the profile is a Widget that can be applied to the UML meta-class, Class. The widgets can be broadly divided into four categories: (i) Container Widgets, (ii) Composite Widgets, (iii) Basic Widgets, and (iv) Interactive Widgets. The container widgets, as the name suggests, can contain other widgets. This includes, for example, the Basic Container, Blinking Container, Mask Container, and Rotation Container. The profile also contains certain composite widgets, representing the widely used standard components, for example, the Altimeter, Air Speed Indicator, and Variometer. We added these composites in the profile to assist the modelers with the most commonly used widgets. These are not directly part of the A661 standard, which deals with the basic, container, and interactive widgets only. Other than these widgets, the profile also contains certain basic widgets (e.g., Line, Diamond, Arrow, and Label) that the modelers can use to model new types of displays. The fourth category of widgets are the interactive widgets with which the pilot can interact. For example, this includes, EditBox, ComboBox, ToggleButton, and CheckButton. Table I presents a few of the concepts that are defined in the profile. The complete profile is downloadable from an open-source repository[2].

C. Profile Instance Model corresponding to CDS

The CDS screens are typically developed in graphical modeling tools, such as VAPS XT and SCADE, that are compatible with the A661 standards. These tools allow exporting of the screen models in XML formats with the details of various widgets.

In the first step of our approach, we model the CDS under test by applying stereotypes of the CDS profile. The CDS model is used to populate an instance model automatically from these screen models. For some cases, the mapping between our profile model and the screen models in XML, especially the composite widgets, is not one-on-one, because the models in XML are typically exported with much finer details than required for our purpose. For such cases, we define a mapping based on the name of the widgets in the screen models. For example, first, the object is identified using the name *AltitudeTape* in the XML model that corresponds to the

[2]https://github.com/hassansartaj/models19

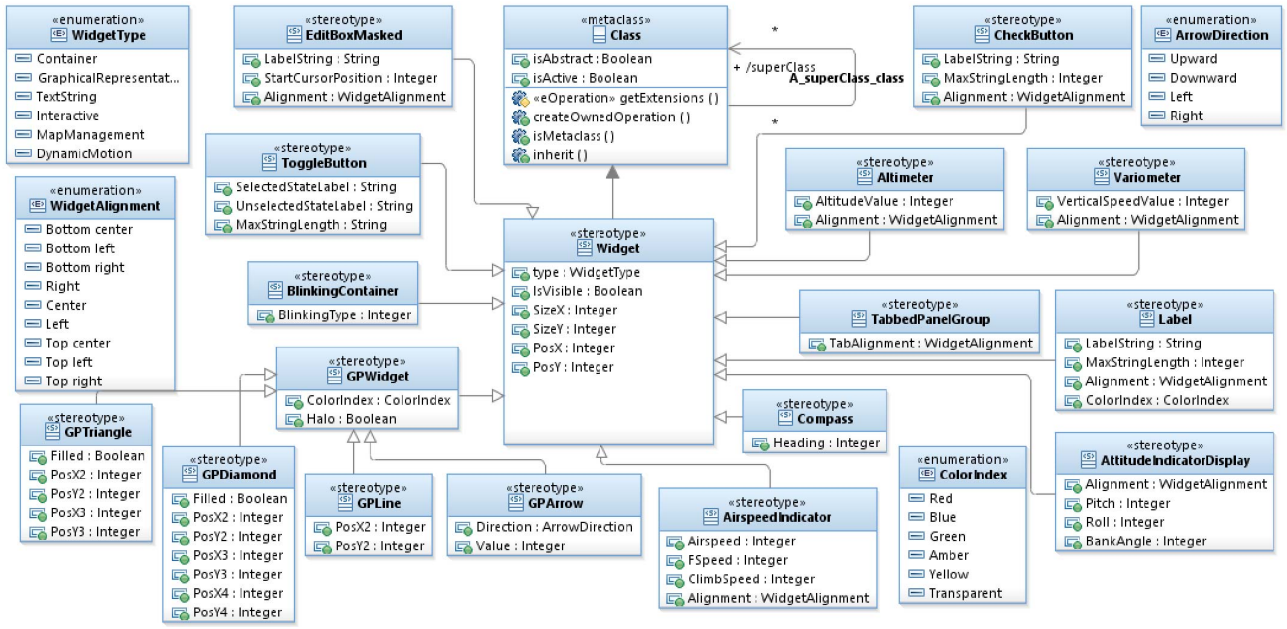

Fig. 3. An excerpt of CDS profile

Altimeter concept of profile. After that, the properties (e.g., position (PosX and PosY), type, size (SizeX and SizeY), and visibility) related to the identified concept are located from the XML model. Finally, the values obtained for all properties of the identified concept are populated in an instance model.

D. Behavioral Modeling of Aircraft

During its flight, an aircraft goes through different states to complete its mission. The information displayed on the CDS screen varies during the flight operations. Similarly, the values of various properties visible on the screens also vary with flight operations. As part of our proposed approach, we require the test engineer to model the behavior of an aircraft that has a direct impact on the CDS. For example, the states that an aircraft may go through, including Taxiing, Landing, TakeOff, and Cruise. We use UML state machine diagrams that are defined as part of Unified Modeling Language (UML) for describing the event-driven behavior of software systems [16].

Fig. 4 shows a reference state machine of an aircraft flight phases. The state machine covers the states of the aircraft flight. The state *Standing* refers to the state of the aircraft in which it is not moving. It consists of two sub-states. First is the *Idle* in which the aircraft is standing idle and all engines are powered off. The second sub-state is *Running* during which the engines are powered on. *ParkingBreakOff()* and *IncreaseThrottle()* are the main events on the transition that allows the aircraft to go into the *Taxiing* state. *Pushback* is a state in which the aircraft needs to be pushed back to move away from the parking stand. *Taxiing* refers to the movement of the aircraft on the runway before takeoff or after landing. It has a sub-state machine for three different turning states (i.e., *Straight*, *TurningLeft*, and *TurningRight*) as shown in Fig. 5. The transition *IncreaseElevation()* takes the aircraft from *Taxiing* to *TakeOff*. *TakeOff* refers to the phase of flight that allows the aircraft to go through a transition from taxiing to flying in the air. When the event *IncreaseElevation()* triggers, the aircraft takes an initial climb and reaches a specific altitude that is mentioned in the guard condition. This phase of the flight is represented by *Climb* state.

After the climbing phase of the aircraft, the aircraft reaches a specific altitude at which it cruises with constant airspeed and altitude. It refers to the *Cruise* in the state diagram. The *Descent* is the phase of the flight in which the aircraft decreases its altitude. The three flight phases i.e., *Climb*, *Cruise*, and *Descent* involve three different types of turns as shown in Fig. 5. *StraightAndLevel* is the phase of the flight in which the aircraft maintain altitude for straight and level flight. During the flight, the aircraft can fly in *Autopilot* mode when the event *SetAPModeOn()* triggers. The autopilot can take the aircraft through four different states i.e., *StraightAndLevel*, *Climb*, *Cruise*, and *Descent*. Therefore, the *Flying* state has two orthogonal states, one for autopilot mode and the other for the four flight phases. The *Approach* and *Landing* are the two last phases of the flight in which the aircraft prepares to land by reducing its altitude and airspeed. The events such as *DecreaseElevation()*, *DecreaseAirspeed()* trigger to go through the transition from *Descent* to *Approach* state and from *Approach* to the *Landing* state.

We provide this reference state machine as support for test engineers. The state machine can be reused by the test engineers and may also be modified. During all the states defined in the state machine diagram, the state values of the CDS widgets and all the relevant information associated

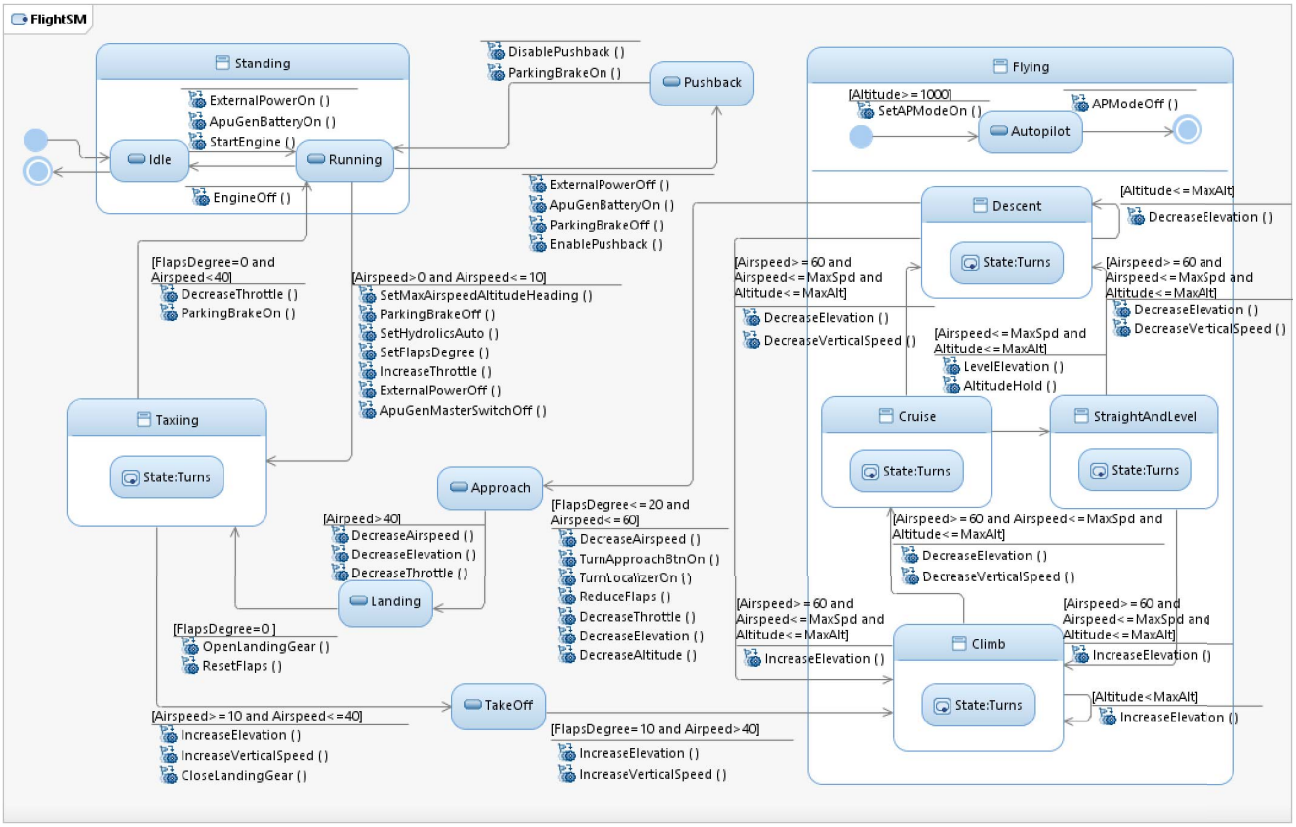

Fig. 4. UML state machine representing the aircraft flight behavior

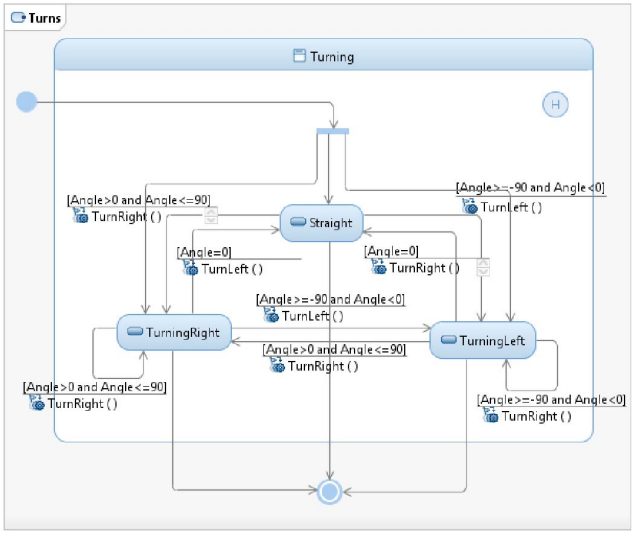

Fig. 5. Aircraft flight sub-state machine for turns

with those widgets changes frequently. This change in the state of the widgets and the associated information occurs because of the external events that are triggered by the aircraft crew in order to carry out flight operations. For instance during the approach, by retracting the flaps the airspeed of the aircraft reduces and the airspeed tape shows the corresponding decrease in the airspeed. Similarly, while landing, when the aircraft reaches at an altitude of 2500 feet, a digital radio altimeter appears on Attitude Indicator Display. This is an example of the change in the state of the widget from invisible to visible based on a specific constraint.

After the flight model is ready, the test engineer is required to model the constraints on the various elements of the CDS corresponding to the states of the aircraft. For this purpose, we use Object Constraint Language (OCL) to specify constraints on the states of our state machine. OCL is a textual language that allows specifying constraints on models [17], [18]. All the constraints are written in the context of an Aircraft class containing the stereotype «Aircraft». These OCL constraints act as an oracle during testing and provide the expected values for the various widgets of the CDS.

E. Testing CDS

In the following, we discuss the proposed strategy for automated testing of cockpit display systems (CDS) of aircraft. A test case in our context is the evaluation of the properties of widgets being displayed on CDS by taking the aircraft into different states. For example, a test case can be taking

the aircraft to a particular altitude and evaluating whether the corresponding values in the Altimeter are updated or not. As per the current industrial practice, taking the aircraft into different states is done with by using simulators. The testers have pre-written test scripts that are loaded into the simulators to execute different scenarios. Following we discuss the steps related to test case generation, test execution, and test evaluation.

1) Test Case Generation: We use the aircraft flight state machine developed in previous steps to generate the test cases. The test paths are obtained using the well-known strategy of achieving round-trip coverage by generating a transition tree from the state machine [16]. The round-trip strategy traverses a state machine to generate end-to-end paths by removing the cycles of the state machine. The approach has been widely used in literature for generating tests from UML state machines [19], [20].

2) Test Case Execution: The first step in test execution is reading the test cases (paths) and taking the aircraft to the desired state by executing the pre-written scripts for the simulator. For example, to take the system into *Taxiing* state, the corresponding script is executed on the flight simulator. An example of a JSBSim script to set throttle value is shown in Listing 1. The name specifies the property for throttle of the engine using the command fcs/throttle-cmd-norm[0]. The range of the value is from 0 to 1.0. The script shown in Listing 1 will be executed after ten seconds of the start of the simulation, and it will set the value of throttle to 0.15 (15%) to start taxiing on the runway.

Listing 1. A script for setting throttle value in a flight simulator

```
<event name="SetThrottle">
  <condition>
    sim-time-sec >= 10.0
  </condition>
  <set name="fcs/throttle-cmd-norm[0]" value="0.15"/>
  <set name="fcs/throttle-cmd-norm[1]" value="0.15"/>
  <set name="fcs/throttle-cmd-norm[2]" value="0.15"/>
  <set name="fcs/throttle-cmd-norm[3]" value="0.15"/>
</event>
```

During the flight simulation, the flight data is sent to the different CDS elements for display. For example, the value of the current altitude of the aircraft is sent to the altimeter in CDS. On reaching a state during simulation, a number of screenshots of the CDS are taken with an interval of one second. The screenshots are to be processed later using image processing techniques. There is no need to perform the image analysis during test case execution, therefore we store the screenshots and process these during the test evaluation phase.

3) Test Evaluation: Once the test execution is completed, the information from each image is automatically extracted. The values of positions obtained from the instance model are used to identify the exact position. We use the properties x-axis, y-axis, x-size, and y-size to identify the location of the required information on the screen. We crop that part into an image using computer vision software and feed it into the optical character recognition software to extract information from the image. This provides us the exact value for the

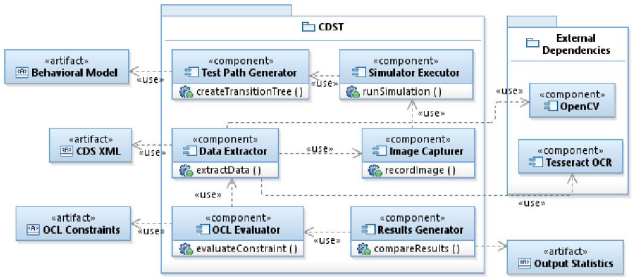

Fig. 6. Component diagram of the CDS testing tool

widget. If a widget is not identified in an image, the testers are asked to tag the various widgets on the screenshot. In our case, most of the widgets we are testing have fixed positions on the screens (which is also the common case). Once a widget is identified, the exact values in the instance model are populated by using an Optical Character Recognition (OCR) tool (in our case Tesseract [21]). When this process is completed we get multiple instance models of the CDS profile-based CDS model during different aircraft flight states. All this information is populated as values of the instance models.

Next, we compare the expected result with the actual results. The expected results are available as OCL constraints. Once a test case is executed, on every state, the corresponding OCL constraints are evaluated. If a constraint results in a *false* outcome, a test case is considered to have detected a potential bug.

IV. Tool Support

In this section, we discuss the tool developed to automate the testing of CDS. The tool consists of six major components, (i) Test Path Generator, (ii) Simulator Executor, (iii) Image Capturer, (iv) Data Extractor, (v) OCL Evaluator, and (vi) Results Generator. Fig. 6 shows the component diagram comprises of all major components of CDS testing tool and the interaction among them. In the following, we discuss each component individually.

A. Test Path Generator

This module takes input a behavioral model in the form of the UML state machine. According to our proposed approach, this module uses the state machine to implement the strategy of achieving round-trip coverage by generating a transition tree. The generated transition tree contains a number of round-trip paths. These paths are used to run the simulation and make the aircraft follow the specified path.

B. Simulator Executor

To simulate the behavior of an aircraft, we use JSBSim [8] flight dynamics model that has been used by many researchers to model the dynamics of flight of an aircraft [22], [23]. For each round-trip path generated by the *Test Path Generator*, a JSBSim script is created to traverse each test path individually. JSBSim is executed for each test script that provides flight data (such as altitude and airspeed) to CDS under test.

C. Image Capturer

This component of the tool is responsible for taking screenshots of the flight simulation during the execution of the flight test paths. The images are captured after the specified interval time. The captured images are stored on the hard disk according to the test path and the aircraft states. These images are used by the *Data Extractor* module to process each image and extract the relevant information.

D. Data Extractor

To extract data from images, this component makes use of an instance model of CDS model developed based on the proposed UML profile. The instance model is populated from the XML produced by a CDS designer tool. The current version of the tool supports the XML generated by VAPS XT. The instance model provides complete information about each widget on CDS. For example, the position (x and y-axis) of the widget, size, and color information. This information is used to identify various widgets in the image. Using the information obtained from the instance model, the subpart of the image containing the target widget is extracted. To extract subpart of the image, an external library OpenCV 3.4.1 [24] is used. After the widget is extracted from the image, the text showing particular information (e.g., altitude, speed) is retrieved. To perform optical character recognition (OCR) in the image, we use an open source tool Tesseract OCR [21].

E. OCL Evaluator

This part of the tool is mainly used to evaluate the OCL constraints against the data extracted from images. First, it takes input OCL constraints corresponding to each state of the aircraft flight. Second, it uses the data obtained from images captured during the simulation and according to the aircraft flight states. The data extracted from images is used to populate the instance model in order to prepare it for OCL constraints evaluation. Finally, the input OCL constraints are evaluated on the instance model. In the case when the data conforms to constraints, OCL evaluator returns *true* and *false* otherwise.

F. Results Generator

In this phase, the evaluation results from *OCL Evaluator* are compiled in the form of a report. The report contains information about covered and uncovered branches of the test path. The report also consists of the information regarding passed and failed OCL constraints and the scenarios (i.e, states) in which the faults are encountered. The report generated by this module helps a test engineer to trace the faults in various CDS widgets.

V. Evaluation

In this section, we apply our proposed approach on an industrial case study for evaluation. First, we provide the details of the case study. Second, we discuss the evaluation setup. Finally, we discuss the evaluation results including the insights and practical applicability.

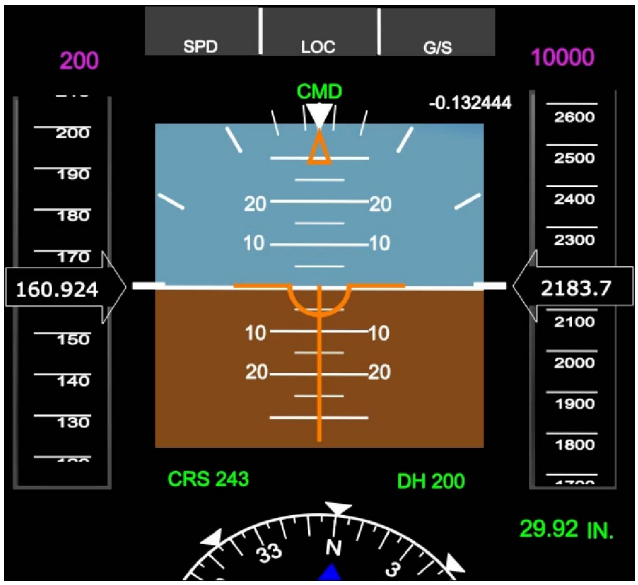

Fig. 7. A screen grab of a Primary Flight Display (PFD)

A. Case Study

The case study used for the evaluation is developed in collaboration with the CDS development team of our industrial partner using VAPS XT [6] tool. The case study comprises of the primary flight display (PFD) for an aircraft as shown in Fig. 7. An excerpt of the XML produced by VAPS XT tool for altimeter part of the PFD is shown in Listing 2. An excerpt of the corresponding instance model for the altimeter, airspeed, and heading indicator part of the PFD is shown in Fig. 8.

Primary Flight Display(PFD) is the main component of an electronic flight instrument system (EFIS). The Primary Flight Display (PFD) is the primary source of flight information for the pilots and displays different type of information like altitude, attitude, airspeed, vertical speed, barometric pressure, and ground speed, etc. Each type of information is shown by a separate graphical widget on the PFD. Thus, PFD is representative of a CDS because it composes the information displayed on individual widgets such as an Altimeter to display altitude, a Vertical Speed Indicator (VSI) to show vertical speed, etc.

Listing 2. An excerpt of the VAPS XT structural model for altimeter part of PFD

```
<object name="AltitudeTape" class="TapeCircular">
  <model>
    <prop name="IsVisible">TRUE</prop>
    <xyprop name="Position" x="131.349" y="-751.194"/>
    <prop name="Value">0</prop>
    <prop name="ValuePerRevolution">10000</prop>
    <structprop name="DisplayArea">
      <field name="Left">-2300</field>
      <field name="Bottom">-5000</field>
      <field name="Right">2300</field>
      <field name="Top">6500</field>
    </structprop>
    <xyprop name="Motion" x="0" y="128571"/>
  </model>
</object>
```

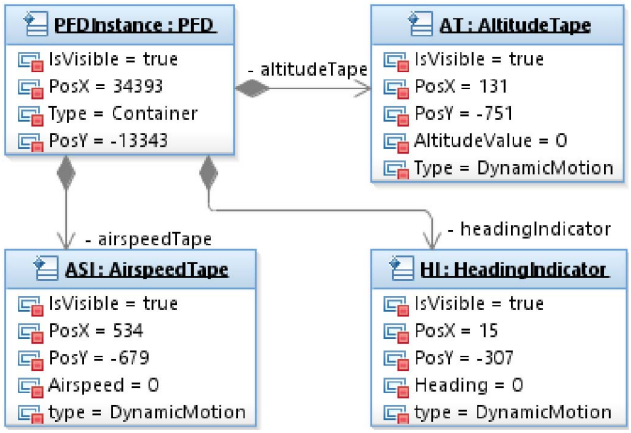

Fig. 8. An excerpt of instance model of the PFD

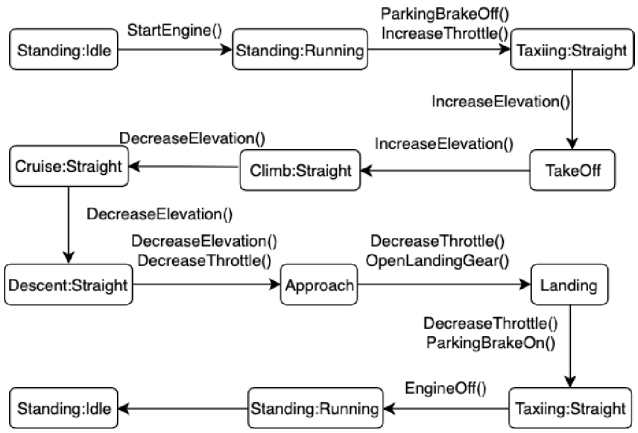

Fig. 9. One test path from the transition tree

TABLE II
MODELING STATISTICS OF THE CASE STUDY

Artifact	Count
No. of states	14
No. of guards	24
No. of transitions	33
No. of classes	18
No. of attributes	173
No. of applied stereotypes	11
No. of constraints	30

A simulation of PFD of an aircraft flying at 2183 feet above sea level (ASL) is shown in Fig. 7. On the left side of the PFD, there is an airspeed tape that shows the airspeed of the aircraft. In Fig. 7 the airspeed is ≈160 knots. On the right-hand side of PFD, there is an altitude tape showing the altitude of the aircraft, i.e., ≈2183 feet above sea level (ASL). The center of the PFD contains the attitude indicator that shows the pitch and roll of the aircraft. Barometric pressure is shown in green color below the altitude tape on the bottom right corner.

Listing 3. An excerpt of the OCL constraints for ground operations

```
1 context Aircraft inv: self.oclIsInState(Standing) and
      self.pfd.airspeedindicator.airSpeed>=0 and self.pfd
      .airspeedindicator.airSpeed<=10 and self.pfd.
      turnIndicator.angle=0
2 context Aircraft inv: self.oclIsInState(Taxiing) and
      self.oclIsInState(TurningLeft) and (self.pfd.
      headingIndicator.angle<0 and self.pfd.
      headingIndicator.angle>=-45)
3 context Aircraft inv: self.oclIsInState(Taxiing) and
      self.oclIsInState(TurningRight) and (self.pfd.
      headingIndicator.angle>0 and self.pfd.
      headingIndicator.angle<=45)
4 context Aircraft inv: self.oclIsInState(Taxiing) and
      self.pfd.airSpeedIndicator.airSpeed>=10 and self.
      pfd.airSpeedIndicator.airSpeed<=60 and self.pfd.
      barometer.airpressure=29.92
```

B. Evaluation Setup

All the structural details of PFD (i.e, the location and relative scales of various widgets and information displayed on them) are represented in a UML class diagram, which is an instance model of our proposed CDS profile. An example of the generated UML class diagram for PFD is shown in Fig. 8. For the behavioral model of an aircraft, we use the reference state machine as shown in Fig. 4. We model the expected properties of the widgets for the aircraft states as OCL constraints.

The modeling statistics for the case study are shown in Table II. The instance model of the profile for PFD consists of 18 classes, 173 attributes, and 11 stereotypes. The state machine that we use for the evaluation contains 14 states, 24 guard conditions, and 33 transitions (as shown in Table II). The constraints for various widgets were identified during different sessions with our industry partner on cockpit-display systems which included the PFD. Listing 3 shows some of the OCL constraints modeled for the states (i.e., *Standing* and *Taxiing*) of an aircraft. The identified constraints were then presented to an avionics and aviation domain expert and any identified corrections and omissions were fixed. The constraint modeling processes resulted in identifying 30 distinct constraints on the various widgets of CDS.

C. Evaluation Procedure

We use the state machine shown in Fig. 4 for generating the test cases. We generate test cases corresponding to the round-trip path coverage criterion [16]. The total number of paths generated using coverage criteria is 494. We select 34 paths that cover all important states required for the complete aircraft flight. One simple test path is shown in Fig. 9. The test case models an end to end scenario of an aircraft flight, from starting its engines to engine shutdown at the end of the flight. To execute the test case, it is necessary to interface with a flight simulator. For PFD case study we use JSBSim [8] to simulate the data for various widgets obtained from the flight dynamics model of Cessna 172 Skyhawk aircraft. For each test path, a JSBSim script is written to execute the simulation. The evaluation statistics are shown in Table IV. The statistics

TABLE III
AVERAGE TIME, TOTAL IMAGES, FAULTY IMAGES, AND FAULTS IDENTIFIED IN EACH STATE FOR PFD

	Standing	Taxiing	TakeOff	Climb	Cruise	Descent	StraightAndLevel	Approach	Landing
Time (m)	9.28	34.87	13.27	153.6	122.67	110.83	26.88	17.95	13.52
Images	557	2092	796	9216	7360	6611	1613	1077	811
Faulty Images	0	475	146	823	492	750	606	621	439
Unique Faults	0	2	3	2	2	3	2	2	2

TABLE IV
EVALUATION STATISTICS OF THE CASE STUDY

Artifact	Value
JSBSim Scripts	34
Instance models	30133
Evaluation Time (m)	1800

include total JSBSim scripts, a total number of instance models and total time (in minutes) spent during the evaluation.

During the flight, the aircraft goes through different states, as modeled in the state machine. At each state during the flight, images are captured after one second and stored with respect to the state. At the end of the simulation, the data from images is extracted and the constraints specified on that state are evaluated. The test case passes if no constraints are violated during the flight, i.e, all displays of PFD function as per the specification.

D. Results and Discussion

In the following, we present the results of the evaluation for an industrial case study of Primary Flight Display (PFD) of an aircraft.

Our automated approach generated 494 test paths in total to test the functionality of PFD. We select 34 paths that cover all important states required for the complete aircraft flight. We identify three major faults in the PFD. One of the identified faults is in the airspeed monitor section of the PFD. During *Descent* state, the airspeed indicator crossed the maximum limit for the airspeed i.e., 200 as shown in Fig. 10. The airspeed indicator tape moved a little ahead when the maximum airspeed limit was reached. As a result, the OCL constraint shown in Listing 4 (C1) failed during the execution as the airspeed was greater than the maximum airspeed.

The second identified fault is the inconsistency between the two different types of turn indicators (Fig. 11). During the execution, one OCL constraint related to the turn angle failed as shown in Listing 4 (C2). The turn angle for one heading indicator (bottom) shows the aircraft is turning right whereas the middle heading indicator shows the angle according to left turn.

The third fault was detected in altimeter tape of PFD. During *TakeOff* state, the aircraft increases the altitude to enter in *Climb* state. The altimeter tape showed constant altitude for a few seconds and then started to increase the value of altitude. In this case, the corresponding OCL constraint failed is shown in Listing 4 (C3).

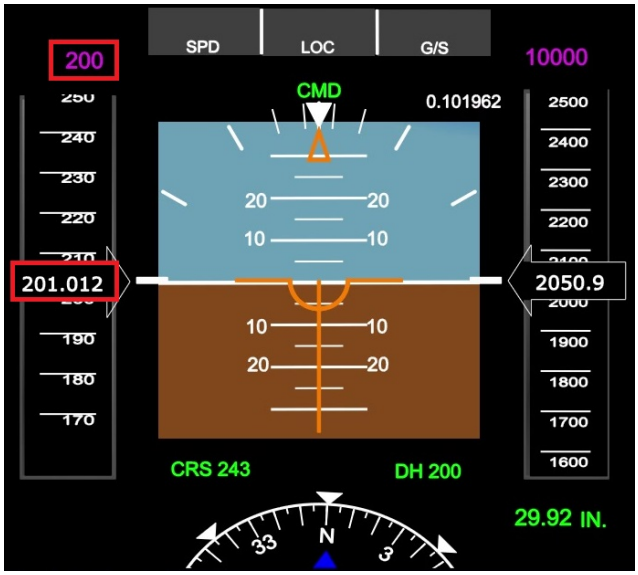

Fig. 10. A bug indicating the airspeed greater than the maximum value

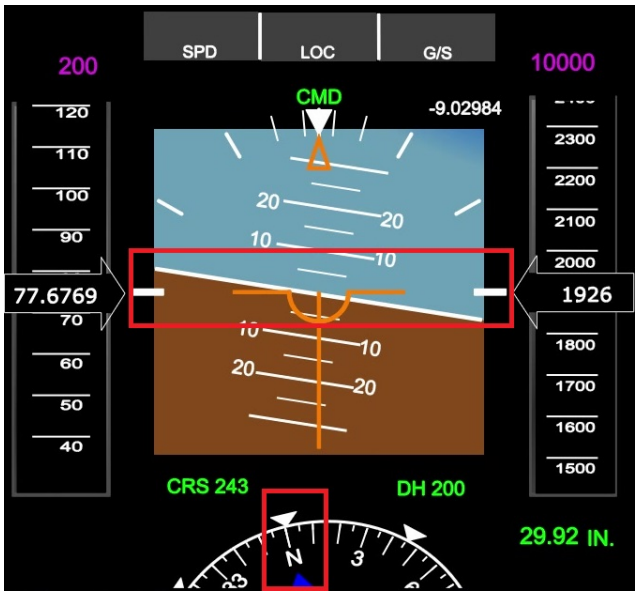

Fig. 11. A bug indicating the inconsistency between the two turn indicators

Table III shows the average time (in minutes) of flight in each state and the number of images captured and processed,

faulty images and the unique faults detected in each state. As can be seen from the table, the generated test cases identified a total of 18 faults. Further inspection of the results showed that in addition to the three faults mentioned above, a further thirteen faults could be attributed to missing functionality in the simulator. This functionality is not implemented in the PFD, however, was reflected in the constraints due to input from the domain expert. This indicates that the approach can also be used to identify missing functionalities.

Listing 4. Some of the violated constraints for PFD

```
C1: context Aircraft inv: self.oclIsInState(Descent) and
      self.pfd.airSpeedIndicator.airSpeed>=60 and self.
    pfd.airSpeedIndicator.airSpeed<=200
C2: context Aircraft inv: self.oclIsInState(Climb) and
    self.oclIsInState(TurningLeft) and (self.pfd.
    headingIndicator.angle<=0 and self.pfd.
    headingIndicator.angle>=-45)
C3: context Aircraft inv: self.oclIsInState(TakeOff) and
      self.pfd.altimeter.altitudeValue>0 and self.pfd.
    altimeter.altitudeValue<=10000
```

Using simulators for testing CDS applications is a common mechanism. Our strategy allows the tester to execute a large number of scenarios and evaluate their results automatically. Though our approach requires familiarity with state machine and OCL constraints modeling. In our experience the avionics engineers are well-versed in developing state machines. Most of the OCL constraints that were written only required a basic knowledge of OCL, however it required a deep understanding of the domain and flight behavior. The proposed profile allowed the domain experts to model constraints using the domain concepts and constructs. The profile also allowed independence from the actual tool that is used to model the CDS.

VI. Limitations

Though the paper provides an automated and systematic approach to CDS of avionics systems, the approach has a few limitations. The first limitation of the proposed approach is that it relies on the test ready behavioral models and constraints written in OCL. Testers have to invest time to get these models and scripts ready.

An important step in our approach is to use image processing to extract relevant information (e.g., text) from various CDS widgets. The prediction accuracy of the OCR engine such as Tesseract [21] poses another limitation to our approach. The accuracy of Tesseract OCR is not always 100% [25]–[27]. To handle this limitation and to enhance the accuracy, we used region-based segmentation and image preprocessing techniques such as noise removal, canny edge detection, and contours finding.

VII. Related Work

The work presented in this paper is the first one to target testing of avionics systems based on the information displayed in CDS. In the following, we discuss the published works that are related to CDS and some relevant works focusing on testing of graphical user interfaces of interactive applications.

Campos *et al.* [28] present an approach to ensure the effectiveness of the interactive applications with automated generation of various validation scenarios using task models. Campos *et al.* [29] improved previous work [28] and generated feasible test scenarios using task models. Catelani *et al.* [30] proposed a technique for the validation testing of the customized TFT-LCD screens that are ready to install in the cockpits of military aircraft. Similarly, Behnken and Salgado [31] present an approach to test display properties of cockpit displays such as color, resolution, position, etc. The benefits of our approach over all the above-mentioned approaches are that our approach is generalizable for modern cockpits and comply with the international standard for CDS (A661 [5]).

A well-known GUI testing tool, GUITAR [32], makes use of event flow graphs by reverse engineering the GUI structure to automatically generate the test cases. In addition to this, some other GUI testing tools such as Android Ripper [33], Amola [34], Orbit [35], etc have also been developed after extensive research. Yeh *et al.* [36] propose Sikuli which is an automated tool to test GUI using screenshots. Chang *et al.* [37] present Sikuli Test with the aim to facilitate testers to write and generate visual test scripts for GUI. Similarly, to perform system-level testing, Alegroth *et al.* [38] proposed a visual GUI testing tool named as JAutomate. Garousi *et al.* [39] conduct an industrial evaluation and highlight the problem with the replay feature of both Sikuli [36] and JAutomate [38]. The main difference of GUI testing approaches with our approach is that the data displayed on CDS is generated by various hardware components (e.g., sensors) of an aircraft based on pilot's interaction and operating environment.

VIII. Conclusion

Testing the avionics of an aircraft is a rigorous process governed by various international standards. As an estimated 70% of all costs of avionics development is spent on testing, software testing being an important part. An important step in testing the user application is to test whether the required information is being displayed correctly on the Cockpit Display Systems (CDS) of an aircraft. The current industrial practice is to test this manually, which is very labor extensive and error-prone. In this paper, we proposed a model-based approach for the automated testing of CDS. We developed a UML profile based on the ARINC 661 standard to model CDS under test. The CDS models developed using graphical modeling tools (e.g., VAPS XT) are automatically converted to an instance of the profile-based model. A test modeler then models the common states of an aircraft during its flight. The modeler also models the constraints on the states of aircraft using Object Constraint Language (OCL). Test cases are generated from the UML state machines, which are then executed using a flight simulator and evaluated using image processing, optical character recognition tools, and OCL evaluator. We apply the approach on an industrial case study of a Primary Flight Display (PFD) developed for an aircraft. The results show that our approach is successful in identifying 18 faults in the PFD, which shows the overall usefulness of the approach.

REFERENCES

[1] S. ARP4754, "Certification considerations for highly-integrated or complex aircraft systems," *SAE, Warrendale, PA*, 1996.

[2] T. K. Ferrell and U. D. Ferrell, "Rtca do-178c/eurocae ed-12c," *Digital Avionics Handbook*, 2017.

[3] Y. Yin, B. Liu, and H. Ni, "Real-time embedded software testing method based on extended finite state machine," *Journal of Systems Engineering and Electronics*, vol. 23, no. 2, pp. 276–285, 2012.

[4] P. Ulbig, D. Müller, C. Torens, C. C. Insaurralde, T. Stripf, and U. Durak, "Flight simulator-based verification for model-based avionics applications on multi-core targets," in *AIAA Scitech 2019 Forum*, 2019, p. 1976.

[5] A. Specification, "661-3 cockpit display system interfaces to user systems," *Aeronautical Radio Inc*, 2007.

[6] VAPS. (2018) Vaps xt. [Online]. Available: https://www.presagis.com/en/product/vaps-xt/

[7] OMG, "Object constraint language specification v2.4," *Object Management Group Inc.*, 2014. [Online]. Available: http://www.omg.org/spec/OCL/2.4/

[8] J. Berndt, "Jsbsim: An open source flight dynamics model in c++," in *AIAA Modeling and Simulation Technologies Conference and Exhibit*, 2004, p. 4923.

[9] I. K. El-Far and J. A. Whittaker, "Model-based software testing," *Encyclopedia of Software Engineering*, 2002.

[10] M. Z. Iqbal, S. Ali, T. Yue, and L. Briand, "Applying uml/marte on industrial projects: challenges, experiences, and guidelines," *Software & Systems Modeling*, vol. 14, no. 4, pp. 1367–1385, 2015.

[11] OMG, "Uml. unified modeling language specification, version 2.5.1," *Object Management Group Inc.*, 2017. [Online]. Available: http://www.omg.org/spec/UML/2.5.1/

[12] Y.-M. Choi and D.-J. Lim, "Automatic feasible transition path generation from uml state chart diagrams using grouping genetic algorithms," *Information and Software Technology*, vol. 94, pp. 38–58, 2018.

[13] S. Ali, M. Z. Iqbal, A. Arcuri, and L. C. Briand, "Generating test data from ocl constraints with search techniques," *IEEE Transactions on Software Engineering*, vol. 39, no. 10, pp. 1376–1402, 2013.

[14] SCADE. (2014) Scade suite. [Online]. Available: http://www.esterel-technologies.com/products/scade-suite/

[15] D. Aviation. (2017) List of arinc661 useful widgets. [Online]. Available: http://j661.sourceforge.net/

[16] R. Binder, *Testing object-oriented systems: models, patterns, and tools*. Addison-Wesley Professional, 2000.

[17] S. Ali, T. Yue, M. Z. Iqbal, and R. K. Panesar-Walawege, "Insights on the use of ocl in diverse industrial applications," in *International Conference on System Analysis and Modeling*. Springer, 2014, pp. 223–238.

[18] M. U. Khan, H. Sartaj, M. Z. Iqbal, M. Usman, and N. Arshad, "Aspectocl: using aspects to ease maintenance of evolving constraint specification," *Empirical Software Engineering*, pp. 1–51, 2019.

[19] S. Iftikhar, M. Z. Iqbal, M. U. Khan, and W. Mahmood, "An automated model based testing approach for platform games," in *Model Driven Engineering Languages and Systems (MODELS), 2015 ACM/IEEE 18th International Conference on*. IEEE, 2015, pp. 426–435.

[20] G. Antoniol, L. C. Briand, M. Di Penta, and Y. Labiche, "A case study using the round-trip strategy for state-based class testing," in *13th International Symposium on Software Reliability Engineering, 2002. Proceedings.* IEEE, 2002, p. 269.

[21] R. Smith, "An overview of the tesseract ocr engine," in *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, vol. 2. IEEE, 2007, pp. 629–633.

[22] O. Cereceda, L. Rolland, and S. O'Young, "Vertical avoidance and recovery analysis of a general aircraft in near mid-air collision scenarios using design and analysis of computer experiments," in *2018 IEEE Canadian Conference on Electrical & Computer Engineering (CCECE)*. IEEE, 2018, pp. 1–4.

[23] A. Hamidani and D. Kunz, "Evaluating the autonomous flying qualities of a simulated variable stability aircraft," in *2018 AIAA Atmospheric Flight Mechanics Conference*, 2018, p. 1019.

[24] G. Bradski and A. Kaehler, "Opencv," *Dr. Dobb's journal of software tools*, vol. 3, 2000.

[25] R. Smith, D. Antonova, and D.-S. Lee, "Adapting the tesseract open source ocr engine for multilingual ocr," in *Proceedings of the International Workshop on Multilingual OCR*. ACM, 2009, p. 1.

[26] C. Patel, A. Patel, and D. Patel, "Optical character recognition by open source ocr tool tesseract: A case study," *International Journal of Computer Applications*, vol. 55, no. 10, 2012.

[27] N. Mor and L. Wolf, "Confidence prediction for lexicon-free ocr," *arXiv preprint arXiv:1805.11161*, 2018.

[28] J. C. Campos, C. Fayollas, C. Martinie, D. Navarre, P. Palanque, and M. Pinto, "Systematic automation of scenario-based testing of user interfaces," in *Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. ACM, 2016, pp. 138–148.

[29] J. C. Campos, C. Fayollas, M. Gonçalves, C. Martinie, D. Navarre, P. Palanque, and M. Pinto, "A more intelligent test case generation approach through task models manipulation," *Proceedings of the ACM on Human-Computer Interaction*, vol. 1, no. 1, p. 9, 2017.

[30] M. Catelani, L. Ciani, M. Venzi, and G. Barile, "Environmental tests and optical measurements in the validation process of tft-lcd for avionics applications," in *Metrology for Aerospace (MetroAeroSpace), 2015 IEEE*. IEEE, 2015, pp. 421–425.

[31] D. Behnken and R. Salgado, "Automated testing of cockpit display visual aspects," *AUTOTESTCON-IEEE-*, vol. 1, no. 1, pp. 551–551, 1992.

[32] B. N. Nguyen, B. Robbins, I. Banerjee, and A. Memon, "Guitar: an innovative tool for automated testing of gui-driven software," *Automated software engineering*, vol. 21, no. 1, pp. 65–105, 2014.

[33] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and A. M. Memon, "Using gui ripping for automated testing of android applications," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2012, pp. 258–261.

[34] Y.-M. Baek and D.-H. Bae, "Automated model-based android gui testing using multi-level gui comparison criteria," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2016, pp. 238–249.

[35] W. Yang, M. R. Prasad, and T. Xie, "A grey-box approach for automated gui-model generation of mobile applications," in *International Conference on Fundamental Approaches to Software Engineering*. Springer, 2013, pp. 250–265.

[36] T. Yeh, T.-H. Chang, and R. C. Miller, "Sikuli: using gui screenshots for search and automation," in *Proceedings of the 22nd annual ACM symposium on User interface software and technology*. ACM, 2009, pp. 183–192.

[37] T.-H. Chang, T. Yeh, and R. C. Miller, "Gui testing using computer vision," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2010, pp. 1535–1544.

[38] E. Alegroth, M. Nass, and H. H. Olsson, "Jautomate: A tool for system- and acceptance-test automation," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*. IEEE, 2013, pp. 439–446.

[39] V. Garousi, W. Afzal, A. Çağlar, İ. B. Işık, B. Baydan, S. Çaylak, A. Z. Boyraz, B. Yolaçan, and K. Herkiloğlu, "Comparing automated visual gui testing tools: an industrial case study," in *Proceedings of the 8th ACM SIGSOFT International Workshop on Automated Software Testing*. ACM, 2017, pp. 21–28.