Design and Implementation: JavaFX Face Detection with Scene Builder and NetBeans IDE

1st Syefrida Yulina
Information System Department
Politeknik Caltex Riau
Pekanbaru, Indonesia
syefrida@pcr.ac.id

***Abstract*—JavaFX face detection is an application widely used to detect and recognize faces in digital images. One of the challenging problems in the image processing is how to develop and design an automatic face recognition application using JavaFX technology. JavaFX is a library of Java that is used to build rich internet applications (RIA) which can run across several platforms such as Desktops, Mobile Devices, TVs, Tablets, etc. Design and implementation of this application applies the concept of model-view-controller using framework Scene Builder and Netbeans IDE. Scene Builder is used as a tool to add components of GUI in the view that can produce document called FXML. Netbeans IDE is an integrated development environment for FXML document editing and maintaining the connection between view and controller. In this research stages of creating JavaFX Face Detection application are started with requirements identification, followed by design of UI components in Scene Graph, integration of the scene builder panels, and then making controller. An application is implemented using the stages of JavaFX.**

Keywords—Face Detection, FXML, Netbeans IDE, Scene Builder, Scene Graph

I. INTRODUCTION

Face detection applications are very necessary in the process of face recognizing. It can gain informations from image or video by using computer algorithms. Many previous studies have developed face detection application using certain programming language but lack of them using Java as a programming language to develop and implement an application based on graphical user interface.

JavaFX is next generation client application platform build on Java. It is a set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications that operate consistently across diverse platforms[1][2]. The interface and implementation of JavaFX application are defined separately from its behavior. By using *model-view-controller* technique, the controllers handle interactions of interface, while the views contain visual attribute/graphical components of interface[3][4]. JavaFX controller are written in Java programming language, and views in the framework are declared in FXML documents[5], where it is written in custom markup language based on the Extensible Markup Language(XML). A controller in JavaFX is used to make GUI interaction in the view class. The views are represented in a tree structure called a scene graph. Scene graph maintains the nodes (graphical components) in a branch node or the leaf node. The first node in tree is called the root node.

Three attributes to make interaction between nodes: 1) *fx:controller* attribute; 2) *fx:id* attribute; and 3) various *event handler* attributes. The *fx:controller* attribute is used to associate the controller with the view by setting the root node's value of the attribute to the name of the controller class. The *fx:id* attributes link field declaration in the controller with their corresponding component instances in the view, to enable programmatic manipulation. *Event handler* attributes assign method declaration in the controller as the recipients of control flow when events are fired by view components. Event handler are required to take a single argument of a type extending javafx.event.Event.

FXML is a textual data format, can be edited in text editor called Scene Builder. JavaFX scene builder enables to quickly design JavaFX application by dragging a UI component from a library of UI components and dropping it into a content view area[6][7][8]. The FXML code for UI layout is automatically generated. Scene Builder can be used as a standalone design tool and can be used in conjuction with Java IDEs. IDE can write, build, and run the controlller source code. JavaFX Scene Builder includes key features such as: a drag and drop WYSIWYG interface, tight interaction with the Netbeans IDE, automatic FXML code generation, live editing anf preview features, access to the complete JavaFX GUI controls library, ability to add custom GUI components to the library, 3D support, support for Rich Text, JavaFX Scene Builder kit, CSS support and cross-platform support.

In this paper, we design an application for face detection with Scene Builder, which integrated with Netbeans IDE. The integration enables Scene Builder by opening FXML document, run the application, and generate controller source file.

II. RESEARCH METHOD

A. Requirements Identification

This research use Netbeans IDE which facilitate JavaFX application development. It builds the connection between views and controller, and the mechanism for source code analysis and manipulation. The link between view and its controller is divided into three parts (Figure 1)

- The *fx:controller* attribute is used to associate the view and controller. It is set as the scene graph's root node. The scene graph for face detection application is shown in figure 2.
- The *fx:id* attributes are declared as component instance in the view from field declaration in the controller. Three *fx:id* are used: btnDetectFace, preProcessImage, originalImage
- *Event handler* attributes is assigned from various component events in the controller. onAction( ) handle the process from source image to detect the face/faces.

Atributes in FXML represent a property of each class instance which it has static property and event handler.

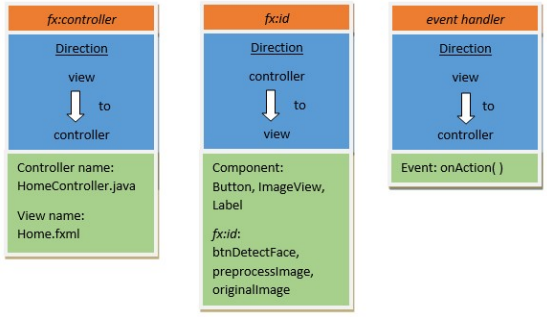

Fig. 1. The Requirements Identification for Face Detection Application

B. Design UI Component in Scene Graph

To build this application, the hierarchical scene graph (Figure 2) should first show the BorderPane layout as a root node, and then it is followed by HBox layout and VBox layout in the next level (child node). Each child node contains UI components such as: ImageView, Label, and Button (leaf node). The implementation of this scene graph can be seen in Figure 5.

BorderPane lays out the children in top, left, right, bottom and center position. The center node in BorderPane is filled the HBox layout which position all its child nodes in horizontal row. VBox is the child nodes of HBox. Each of VBox contains two child nodes: ImageView and Label in first of VBox, two Buttons in the second VBox, and ImageView and Label for the last VBox.

The ImageView component is a node used for displaying original image and result image after face detected. The Button component is a node which can respond to mouse event by implementing an event handler to process the mouse event. The view for this JavaFX face detection is shown in figure 3.

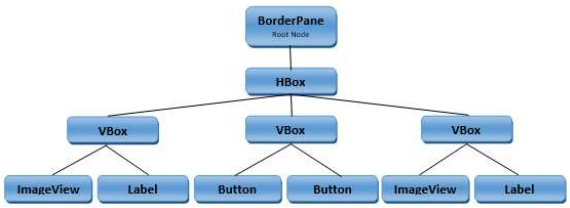

Fig. 2. Scene Graph for UI Components

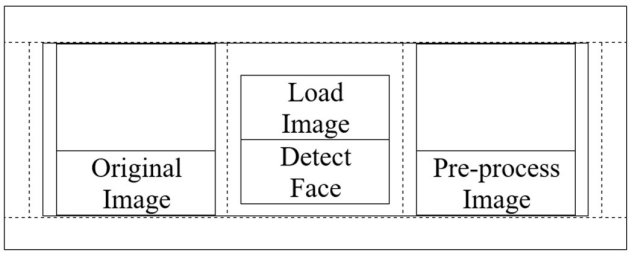

Fig. 3. The View of Application in Scene Builder

C. Integration of the scene builder panels (FXML & View)

Scene builder's library, document and content panels are extracted to Netbeans view in FXML document. Library panel display a collection of the GUI components that are used to build the view. Document panel is comprised of two sub-panels: the Hierarchy and the Controller panel. The Hierarchy panel display the view components in a tree structure. The Controller panel manage the view to its controller, which has an input field for the controller's name, as well as an overview of all the *fx:id* declaration in the view. The content panel display a static preview of view's content will look like when rendered in a running application. The FXML document loads component instances declared in the view into the controller's field declaration.

D. The Controller

Scene builder component implements the connection with controller. The controller input field and event handler (in *fx:id*'s) are declared with the notation @FXML in Java class. This notation integrates scene builder's component to be recognized by its controller. The controller enables field and event handler access through FXML loader by importing javafx.fxml.FXML. The FXML loader is responsible for loading the FXML source file and returning the value of each component in the scene graph.

The input fields such as: origrinalImage and preprocessImage attibutes are declared by importing package javafx.scene.image.ImageView, while btnDetectImage and btnPreprocess attributes import package javafx.scene.control.Button. Event handler utilized to process images to be able to detect faces is onAction( ) to call method loadImage( ) dan detectAndDisplay( ). The method of loadImage( ) requests user to select image file to be processed, and then the method of preProcess( ) will process the images that have been input to be grayscale image by using the library open source version 3.4 from OpenCV. The method of detectAndDisplay( ) will process images that have been input for faces to be able to be detected by using the feature of Haar Cascade Classifier. This feature is obtained from the open source library of OpenCV by importing the package org.opencv.objdetect.CascadeClassifier.

III. RESULT AND DISCUSSION

In this research, JavaFX application to detect faces in images has been designed and implemented by using Scene Builder and Netbeans IDE (shown in Figure 4). The preliminary stage of designing this application is by determining its requirements such as *fx:controller, fx:id*, and *event handler*. What comes next is that every single GUI component needed is designed into a scene graph to view the component structure used in the display. The integration of the display in scene builder uses file FXML which is recognized by controller so that every component can be declared and processed in the programming syntax. In the controller, various libraries required have been imported from JavaFX and library OpenCV.

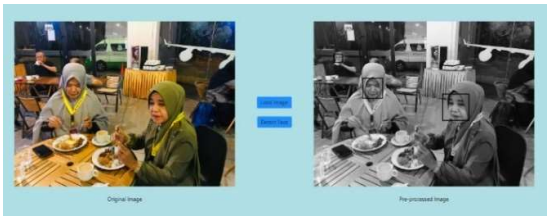

Fig. 4. JavaFX Face Detection Application

The controller contains input field and event handler using notation @FXML, which integrate the component to be recognized. It shown in Figure 5.

```
public class HomeController {

    @FXML
    private ImageView originalImage, preprocessImage;

    @FXML
    private Button btnDetectImage, btnPreprocess;

    protected void init( ) {…}

    @FXML
    protected void loadImage( ) throws IOException {…}

    @FXML
    protected void detectAndDisplay( ) {
        String xmlFile = "D:/haarcascade_frontalface_alt2.xml";
        CascadeClassifier classifier = new CascadeClassifier(xmlFile);

        MatOfRect faces = new MatOfRect( );
        classifier.detectMultiScale(imageGray, faces);
        JOptionPane.showMessageDialog(null, String.format("Detected %s faces", faces.toArray( ).length));

        for (Rect : faces.toArray( )) {
            Imgproc.rectangle(imageGray, new Point(rect.x, rect.y), new Point(rect.x + rect.width, rect.y +
            rect.height), new Scalar(0, 0, 255), 3);}
            preProcess( );
            }

    @FXML
    protected void preProcess( ) {…}

    public void setStage(Stage stage) {…}

    private void updateImageView(ImageView view, Image image) {…}
}
```

Fig. 5. HomeController.java

While controllers can be easy to write event handlers in script, it is preferable to define complex application logic in a compiled using Java language. The *fx:controller* attribute allows a caller to link a "controller" class with an FXML document. It is a compiled class that implements the "code behind".

The view from scene builder is generated to Netbeans in FXML code (Figure 6 and Figure 7). In FXML, all classes are imported such as java.lang package to following processing import the VBox, ImageView and Button classes. The root node assigns the *fx:controller* to connect the view and the controller. The HBox layout as the child node on the level 1 is used to arrange the other series of nodes in a single row, and the VBox layout as the child node on the level 2 is used to arrange the other nodes in single column. The VBox layout contains UI components with their *fx:id*.

Each property for an object in FXML is being set. Two ImageView components have *fx:id* originalImage and fx:id preprocessImage. The *fx:id* original image is then assigned to contain the original image from the file chooser in which *fx:id* preprocessImage is assigned to show the preprocessed image. One button component has *fx:id* btnDetectFace which is declared to process the image and detect the face(s). In document panel, the controller is setting up the value of controller class: HomeController.java.

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.image.*?>
<?import java.lang.*?>
<?import javafx.scene.layout.*?>

<BorderPane maxHeight="-Infinity"
            maxWidth="-Infinity"
            minHeight="-Infinity"
            minWidth="-Infinity"
            xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1"
            fx:controller="projectfriver.HomeController">
  <center>
    <HBox alignment="CENTER" spacing="20.0" BorderPane.alignment="CENTER">
      <children>
        <VBox alignment="CENTER" spacing="20.0">
          <children>
            <ImageView fx:id="originalImage" fitHeight="150.0" fitWidth="200.0" pickOnBounds="true"
preserveRatio="true">
              <VBox.margin>
                <Insets left="30.0" right="30.0" />
              </VBox.margin></ImageView>
            <Label text="Original Image" />
          </children>
          <HBox.margin>
            <Insets />
          </HBox.margin>
        </VBox>
        <VBox alignment="CENTER" spacing="20.0">
          <children>
            <Button mnemonicParsing="false" onAction="#loadImage" text="Load Image" />
            <Button fx:id="btnDetectFace" mnemonicParsing="false" onAction="#detectAndDisplay"
text="Detect Face" />
          </children>
          <HBox.margin>
            <Insets />
          </HBox.margin>
        </VBox>
        <VBox alignment="CENTER" spacing="20.0">
          <children>
            <ImageView fx:id="preprocessImage" fitHeight="150.0" fitWidth="200.0" pickOnBounds="true"
preserveRatio="true" />
            <Label text="Pre-processed Image" />
          </children>
          <HBox.margin>
            <Insets left="30.0" right="30.0" />
          </HBox.margin>
        </VBox>
      </children>
      <BorderPane.margin>
        <Insets bottom="70.0" left="30.0" right="30.0" />
      </BorderPane.margin></HBox>
  </center>
</BorderPane>
```

Fig. 6. Home.FXML

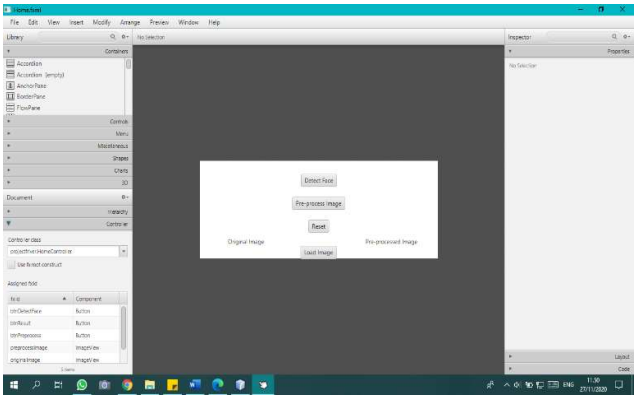

Fig. 7. Home.FXML in Scene Builder

IV. CONCLUSION

In the stage of UI component integration in the display on the controller, there are several things that have to be fulfilled,

such as: assigning fx:controller, fx:id and event handler in UI component must be performed in document panel in Scene Builder so that the display can be recognized by the controller; the use of @FXML notation must be added to the attribute declaration and method to process the input; the FXML document loader should be assigned with the controller's name. JavaFX face detection is an application to detect face using image as an input. It is develop and implement based on the stages: requirements identification, design of UI components in Scene Graph which simplify developer to design based on hierarchy of components, integration of the scene builder panels, and controller.

REFERENCES

[1] O. Corporation, "Introduction to FXML," 2012. [Online]. Available: https://docs.oracle.com/javafx/2/api/javafx/fxml/doc-files/introduction_to_fxml.html [Accessed: 20-Mar-2020].

[2] O. Corporation, "JavaFX: Getting Started with JavaFX," 2014. [Online]. Available: https://docs.oracle.com/javase/8/javafx/get-started-tutorial/index.html [Accessed: 20-Mar-2020].

[3] G. Kruk, O. Alves, L. Molinari, and E. Roux, "Best practices for efficient development of JavaFX applications," JACoWPublishing, pp. 1078–1083, 2018, doi: 10.18429/JACoW-ICALEPCS2017-THAPL02.

[4] I. Science, "Comparison and implementation of graph visualization algorithms using JavaFX," 2016.

[5] A. Pomaroli, JavaFX Programming CookBook. Hot Recipes for JavaFX Development. Exelixis Media, 2016.

[6] H. Qluon, "Scene Builder Documentation," 2016. [Online]. Available: https://docs.gluonhq.com/scenebuilder/ [Accessed: 20-Mar-2020].

[7] T. Ask, "Tobias Ask Master ' s thesis sb4e : an open source integration of the Scene Builder GUI editor into the Eclipse IDE Facilitating JavaFX application development with an extension to the IDE," no. June, 2019.

[8] A. Alkhars and W. Mahmoud, "Cross-Platform Desktop Development (JavaFX vs. Electron)," 2017.