August 2021

TESTING LIBRARY FOR SMART DRIVING COMPANION APPLICATIONS

Rafael Lima

Billy Lam

TESTING LIBRARY FOR SMART DRIVING COMPANION APPLICATIONS

ABSTRACT

This publication describes systems and techniques for providing a testing library for smart driving companion applications. A smart driving companion application may be an application that executes at a mobile computing device, such as a smartphone, to project a graphical user interface (GUI) to the head unit of a vehicle, such as by projecting its GUI to a display device of an infotainment system of a car or truck. A developer may write a smart driving companion application that uses a smart driving companion library that is specific to smart driving companion applications.

A developer of a smart driving companion application can use a testing library to test the use of such a smart driving companion library. The testing library may implement the internal logic of the methods and classes of the smart driving companion library, and the developer may use the testing library to test whether the application behaves correctly in response to user input and as the application moves through different states of the application's lifecycle.

DESCRIPTION

A vehicle, such as a car or truck, may include a so-called "head unit" that presents a GUI by which to control vehicle systems, such as a heating, ventilation, and air conditioning (HVAC) system, a lighting system (for controlling interior and/or exterior lights), an infotainment system, a seating system (for controlling a position of a driver and/or passenger seat), etc. The GUI may be presented via a console, such as an in-vehicle display.

A mobile computing device can connect to the head unit of the vehicle via a wired (e.g., Universal Serial Bus) or wireless (e.g., BLUETOOTH, WIFI, etc.) connection. The mobile computing device may execute smart driving companion applications to project the GUIs of the applications to the in-vehicle display of the head unit so that the driver or passengers of the vehicle may view and/or interact with the GUIs of the smart driving companion applications projected to the in-vehicle display. Such smart driving companion applications may make use of a smart driving companion library that enable the smart driving companion applications to project GUIs of the applications to the in-vehicle display of the head unit. Examples of such smart driving companion applications may include navigation applications that projects a map GUI for providing turn-by-turn directions to help the driver of the vehicle navigate to a destination, a music streaming application that plays music through the infotainment system of the vehicle, a phone application that provides handsfree calling functionality for the driver of the vehicle, and the like.

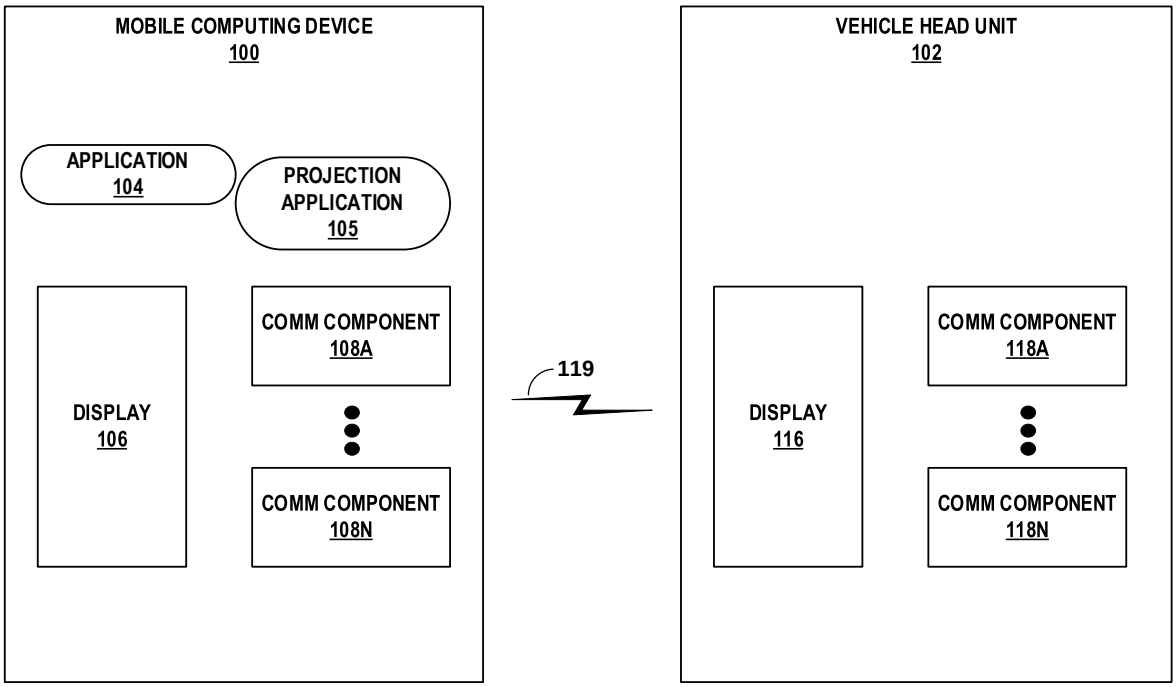

FIG. 1

FIG. 1 is a block diagram that shows an example mobile computing device 100 that can connect to an example vehicle head unit 102. As shown in FIG. 1, mobile computing device 100 can connect to vehicle head unit 102 via connection 119, which may be a wired or wireless connection as discussed above. Mobile computing device 100 includes application 104, projection application 105, display 106, and communication components 108A-108N. Application 104 is an example of a smart driving companion application described in this publication. Projection application 105 may represent an application that facilitates projection of the GUI of application 104 to display 116 of vehicle head unit 102 (and providing a bridge between approved additional applications 104 – e.g., for music streaming, navigation, telephone, notifications, etc. – and projection of GUIs for these applications 104). Display 106 of mobile computing device 100 may represent a presence-sensitive display that functions as an input device and as an output device. Communication components 108A-108N may include wired and/or wireless communication devices capable of transmitting and/or receiving communication signals to and from vehicle head unit 102 to establish connection 119 with vehicle head unit 102.

Similarly, vehicle head unit 102 includes display 116 and communication components 108A-108N. Display 116 of vehicle head unit 102 may represent a presence-sensitive display that functions as an input device and as an output device. Communication components 118A-118N may include wired and/or wireless communication devices capable of transmitting and/or receiving communication signals to and from mobile computing device 100 to establish connection 119 with mobile computing device 100.

An application may use a smart driving companion library to function as a smart driving companion application, such as application 104 shown in FIG. 1, that projects a GUI to the in-vehicle display (e.g., display 116 of vehicle head unit 102). That is, the application may call

functions of the library, implement classes specified by the library, and the like to enable the application to control the GUI projected to the in-vehicle display. For example, the user interface of a smart driving companion application is represented by a graph of model objects that can be arranged together in different ways as allowed by a template that acts as a root in those graphs. Models include the information, such as in the form of text and images, to be displayed to the user, as well as attributes to configure aspects of the visual appearance of such information (e.g., text colors or image sizes).

Screen is a class provided by the library that the application may implement to manage the user interface presented by the in-vehicle display. A Screen has a lifecycle and provides the mechanism for the application to send the template to the in-vehicle display when the screen is visible. Screen instances can also be pushed and popped to and from a Screen stack, which ensures they adhere to any template flow restrictions.

A CarAppService is an abstract Service class provided by the library that the application may implement and export in order to be discovered and managed by the Host. The CarAppService is responsible for validating that a host connection can be trusted and subsequently providing Session for the Application.

A Session is an abstract class that the application may implement to serve as the entry point to display information at the in-vehicle display. A Session may have a lifecycle that informs the current state of the application's user interface at the in-vehicle display, such as when the user interface of the application is visible at the in-vehicle display or is hidden..

A mobile computing device may execute a back-end component (e.g., projection application 105), referred to herein as the host, that implements the functionality offered by the library's application programming interfaces (APIs) in order for the application to function as a

smart driving companion application that projects a GUI to the in-vehicle display. The responsibilities of the host range from discovering the application and managing the application's lifecycle, to converting models defined by the application into views and notifying the application of user interactions with the application's user interface.

A developer of an application for a mobile computing device may create unit tests to test the functionality of the application. Such unit tests may include local unit tests, which may execute at a desktop or laptop computer rather than at the mobile computing device. For example, such unit tests may execute in a Java Virtual Machine (JVM) at a desktop or laptop computer. If the application depends on frameworks provided by the operating system of the mobile computing device, the developer may use a testing framework, such as Roboelectric, that executes the internal logic of such frameworks, to create local unit tests for the application that executes at a desktop or laptop computer. However, current testing frameworks, such as Roboelectric, may not support testing of smart driving companion applications that depend on a smart driving companion library because such testing frameworks may not implement the internal logic of the smart driving companion library on which smart driving companion applications may depend.

As such, this publication describes a smart driving companion testing library may implement the internal logic of the methods and classes of a smart driving companion library that is used to build smart driving companion applications that executes at mobile computing devices. A developer may use the smart driving companion testing library to enable the developer to create and execute local unit tests to test the functionality of a smart driving companion application at a desktop or laptop computer rather than at the mobile computing device. For example, the developer may use the smart driving companion testing library to create and

execute local unit tests to test whether the application behaves correctly in response to user input and as the application moves through different states of the application's lifecycle.

In some examples, the smart driving companion testing library may allow a smart driving companion application to set up a Screen object for testing and provide APIs to allow moving a Screen through different lifecycle states. In some examples, the smart driving companion testing library may enable the developer to test the CarAppService object of the smart driving companion application and may provide APIs to allow the smart driving companion application to move the CarAppService object through different lifecycle states.

In some examples, the smart driving companion testing library may enable the developer to retrieve Template objects from Screen objects of the smart driving companion application and to access the internal values of the Template objects during testing of the smart driving companion application. Such internal values may be in the form of a primitive, such as a string or an integer, or in the form of a controller of the internal value. Returning controllers for the internal values may enable the developer to chain getter method calls when writing the unit tests.

In some examples, the smart driving companion testing library may provide APIs for performing user actions, such as user interactions with the user interface of the smart driving companion application. The developer may use such APIs to test the behavior of the application in response to user actions. For example, the developer may test that proper callbacks take place in response to user actions and/or may monitor internal state changes of the application in response to user actions.

The smart driving companion testing library may enable the developer to test the behavior of the smart driving companion application given specific user input and/or state changes to test whether the application performs certain actions in response to a specific user

input and/or state change. For example, the smart driving companion testing library may enable the developer to test whether the application is displaying a Toast, which is a small pop-up user interface element that provides simple feedback about an operation, whether the application is requesting a Surface, and/or whether the application is pushing a new Screen.

Because the application may call the APIs of different managers of each of these actions, the smart driving companion testing library may implement corresponding test classes of such managers that can be used by the developer to test whether the application performs these actions in response to a specific user input and/or state change. The smart driving companion testing library may implement the corresponding test classes of such managers as static singletons that track the last actions taken on each of the classes and store the values passed to the classes, so that the developer may track these values for the purposes of testing the application.

In some examples, the smart driving companion testing library may also introduce a class that enables testing of actions that could be performed by the host, such as setting the night mode of the in-vehicle display.

It is noted that the techniques of this disclosure may be combined with any other suitable technique or combination of techniques. As one example, the techniques of this disclosure may be combined with the techniques described in U.S. Patent Application Publication No. 2019/0179734 A1. As another example the techniques of this disclosure may be combined with the techniques described in “TestAppManager”, 15 October, 2020, available online at https://developer.android.com/reference/com/google/android/libraries/car/app/testing/TestAppManager. As another example the techniques of this disclosure may be combined with the techniques described in Walton, Philip, “Page Lifecycle API”, 28 May, 2020, available online at https://developers.google.com/web/updates/2018/07/page-lifecycle-api.