Testing the Modern Mobile World

Building Complex End-to-end Mobile Tests

Ang Li
Software Engineer
Google Inc.

The Expanding Landscape of the Modern Mobile World

...and Its Testing Challenges

Google

Multi-device Interaction

- Conference calls
- Money transfer
  - Google Pay
- Peer-to-peer file transfer
  - Files by Google
- Proximity detection
  - Instant tethering
- Sync state across multiple devices
  - Phone to TV login
- Multi-screen gameplay
  - Stadia

Challenge: Controlling multiple devices with interlocking steps in a test.

Non-mobile Devices

- Internet-of-Things (IOT)
  - Thermometers
  - Security cameras
- VR Headsets
  - Google Daydream
- Communication infrastructure
  - Project Loon
- Equipment for altering physical environment
  - Attenuators for RF signals
  - Robotic components for physical movements

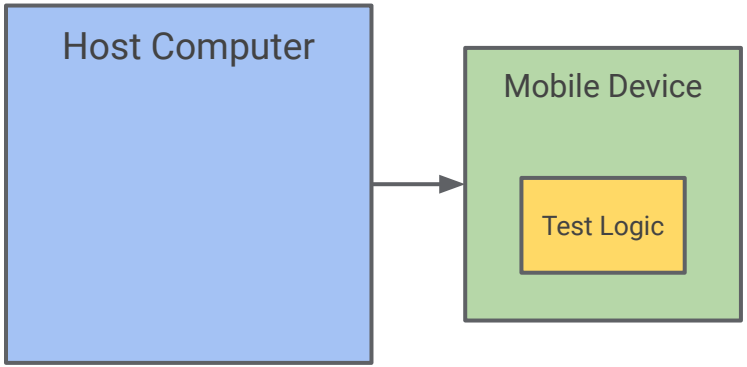
Challenge: Controlling a variety of non-mobile devices and equipments with conventional mobile devices in a test

General Requirements

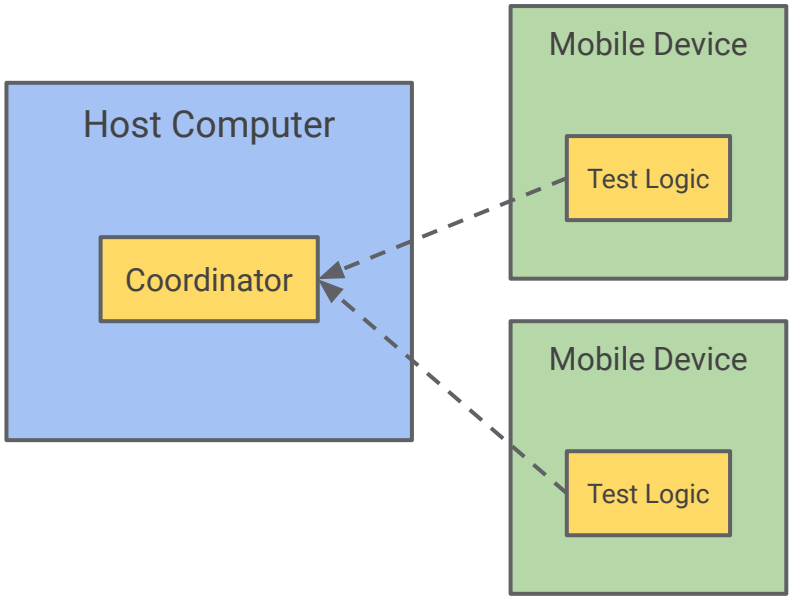
- Full control of the mobile devices
  - Everything the single-device tests do
  - External-driven actions, like reading sys logs, rebooting the device etc
- Coordinate multiple devices to create user scenarios
  - Messaging between two clients
  - Adjust thermostat via mobile app
- Supports non-mobile devices
  - Other device types like wearables and IOT devices
  - Test instruments like power meters, attenuators
- **Easy to write and debug tests**

Common Architecture of Mobile Test Frameworks

Single Mobile Device, Device-driven



Distributed



Problems

- Test logic has little to no visibility outside of the Device Under Test (DUT)
  - Tests are affected by problems of the DUT
  - Difficult to control and coordinate multiple devices..
- UI centric
  - Limited to no option to conduct a test without UI
- Designed for unit tests
  - Often assuming single-app tests
- Difficult to adopt for non-mobile devices

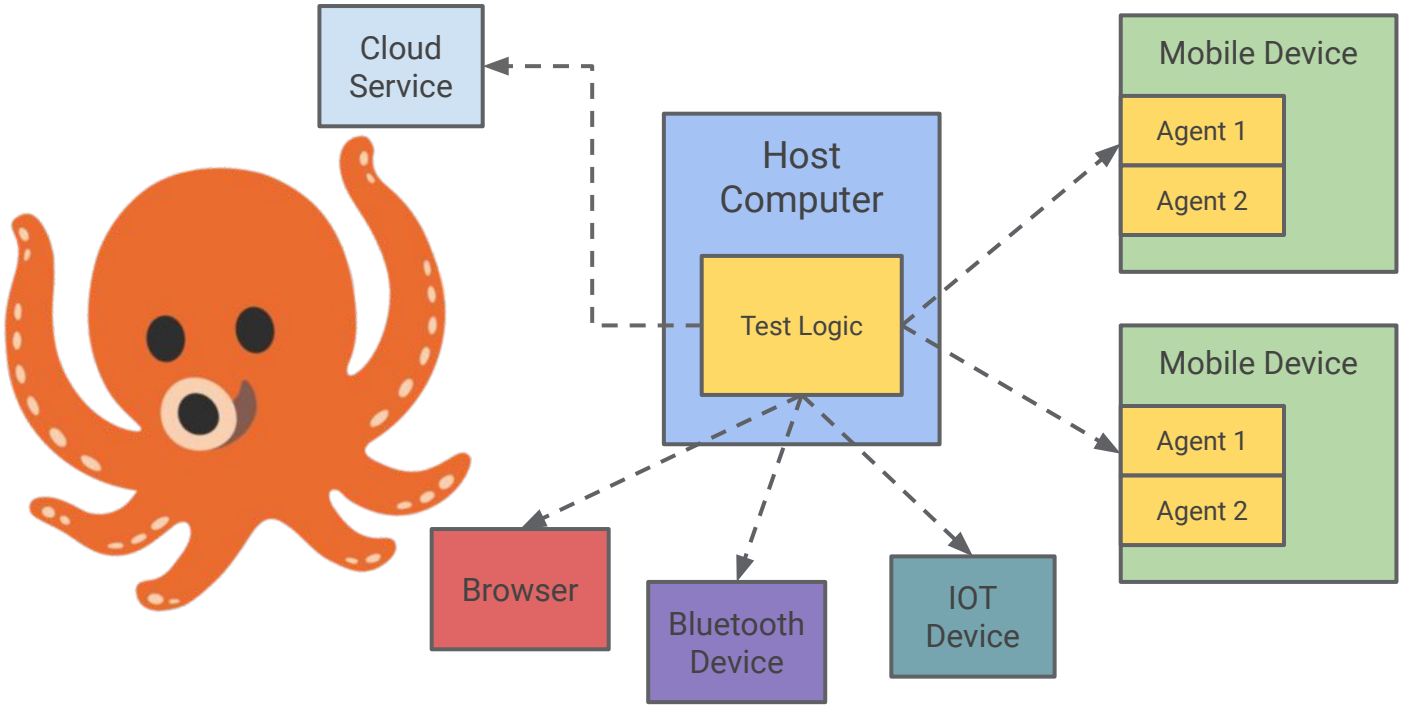
Solutions

Host-driven Generic Controller Architecture

with two key design axioms

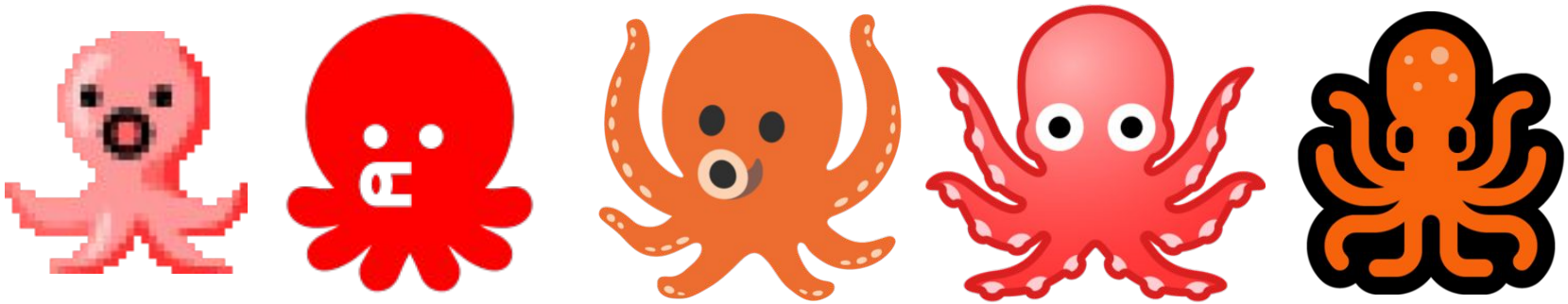
- The test logic shall reside at a central location, often a host computer
- The framework shall assume generic controllers as components instead of mobile devices

Example Logic Diagram



Advantages

- Test logic is a streamlined piece of code at one place
  - Easy to understand and debug
- Test can access multiple test components
- Test logic can coordinate actions to simulate a user scenario
- Test can operate non-mobile device components
  - Plug any device types you'd like into your mobile tests



Tools Built for the Task

- Mobly
  - A test framework designed for creating host-driven tests
  - No assumption on the type of the test components
- Mobly Snippet Library
  - A RPC library for host logic to communicate with test components
  - Allow users to build custom device-side steps to be triggered by host-side main logic
- Mobly Bundled Snippets for Android
  - A set of pre-implemented snippets for basic Android operations
  - Operations such as:
    - Make a toast
    - Enable Bluetooth
    - Add a Google account

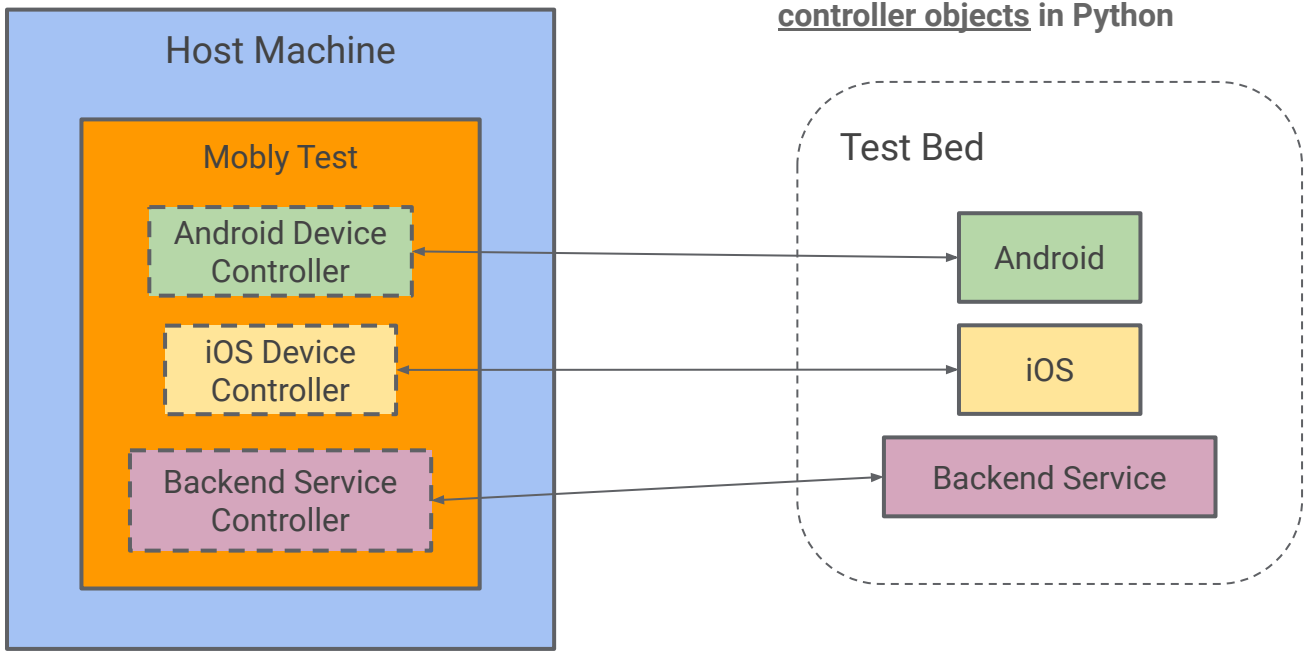
Mobly Test Framework

A Python test runner for creating complex end-to-end mobile tests

- A Mobly test operates on a collection of controllers
  - **Test Bed** - a collection of components used in the test (Phones, services, equipment etc)
  - **Controllers** - objects in Python script representing components in the test bed.
- Flexible and pluggable
- Open source

Testbed + Controller Structure

One-to-one mapping between the actual devices/services in the test bed and controller objects in Python



Controller "Interface"

A loosely defined module level "interface" for declaring arbitrary test components

- `create`
  - The factory function that translates configs into objects
  - Config schema is entirely up to the controller owner
- `destroy`
  - Tear down the controller objects generated by `create`
- `get_info`
  - Retrieve any useful info from the controller after the test run

Example Controller - AndroidDevice

Distributed as part of the Mobly pip package

- Full ADB access
  - AndroidDevice#adb
  - ad.adb.shell(['ls', '/sdcard'])
- Long-running service management
  - logcat
  - supports custom service
- Designed to handle most active Android versions
- Client for Mobly snippets

Designed for Complex Tests

Similar to standard unit test, with additional features to accommodate more complex tests.

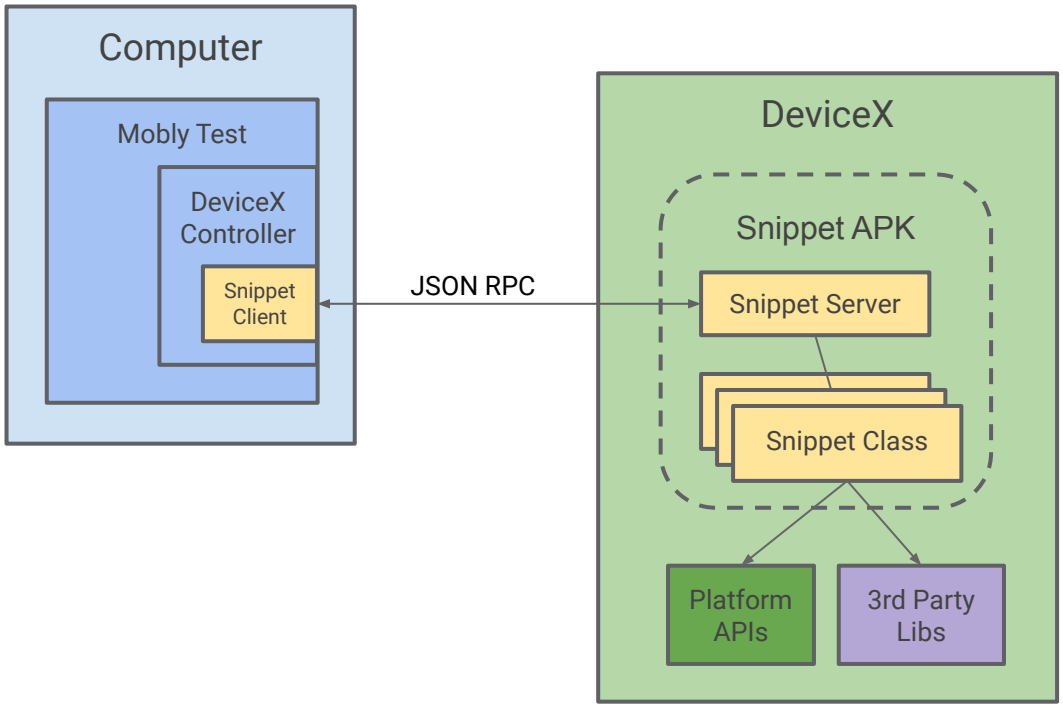
- Define multiple failure points in a single test
  - `expects` APIs
- Rich reporting structure
  - Add extra info in assertions
  - Add custom sections specific to your testing operation
- Conditional stages triggered by test status
  - `on_fail` is commonly used for debug info collection and recovery

Mobly Snippet Library

A standardized RPC protocol for interacting with a device in Mobly tests.

- All the power of a single device test
  - Android: Instrumentation test
  - iOS: XCTest, XCUITest
  - Common libraries like Espresso/UI Automator
- Call the native platform APIs
  - Java, Swift, Objective-C etc
- Synchronous and Asynchronous calls

Mobly Snippet Library



Mobly Snippet Library - Android Java

```
package com.mypackage.testing.snippets.example;

public class ExampleSnippet implements Snippet {
  public ExampleSnippet(Context context) {}

  @Rpc(description='Returns a string greeting the user by name.')
  public String sayHello(String name) {
    return "Hello, " + name + "!";
  }

}
```

Mobly Snippet Library - Invoking from Python

```
from mobly import base_test
from mobly.controllers import android_device

class ExampleTest(base_test.BaseTestClass):

  def setup_class(self):
    self.ad = self.register_controller(android_device)[0]
    self.ad.load_snippets(name='snippet',
                          package='com.mypackage.testing.snippets.example')

  def test_foo(self):
    foo = self.ad.snippet.sayHello('Jeff')  # `foo` is "Hello, Jeff!"
```

Mobly Snippet Library - iOS

```objc
#import <Mobly/Mobly.h>

@interface ExampleSnippet: NSObject
@end

@implementation ExampleSnippet
+ (void)load {
  NSArray *methods = [MBLMethod methodsWithReference:[ExampleSnippet class]
                                           selectors:@selector(sayHello:), nil];
  [MBLRegistry.sharedInstance registerRPCMethods:methods];
}

+ (NSString *)sayHello:(NSString *)name {
  return [NSString stringWithFormat:@"Hello, %@!", name];
}
@end
```

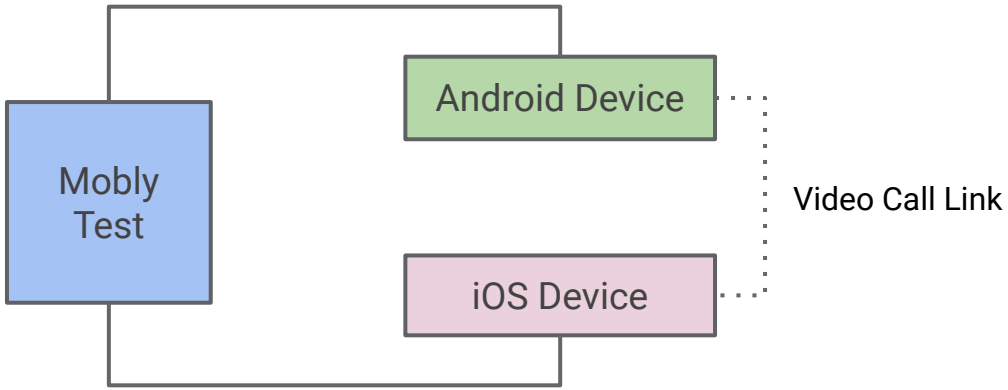
Sample Code & Use Cases

Example Test: Cross-platform Video Call

Making a phone call between one Android and one iOS devices.

- Assuming we already have phone call related snippets implemented.

Cross-platform Video Call



Test Bed Config

```
TestBeds:
  - Name: SampleTestBed
    Controllers:
        AndroidDevice:
          - serial: ABCDEFG1234567
          - phone_number: 4150007890
        IosDevice:
          - udid: 7654321-asdfghjkl
          - phone_number: 4150001234
```

Acquiring Controller Objects

```
from mobly import base_test
from mobly import expects
from mobly.controllers import android_device
from mobly.controllers import ios_device

class CallTest(base_test.BaseTestClass):

  def setup_class(self):
    self.android = self.register_controller(android_device)[0]
    self.iphone = self.register_controller(ios_device)[0]
```

Test Logic

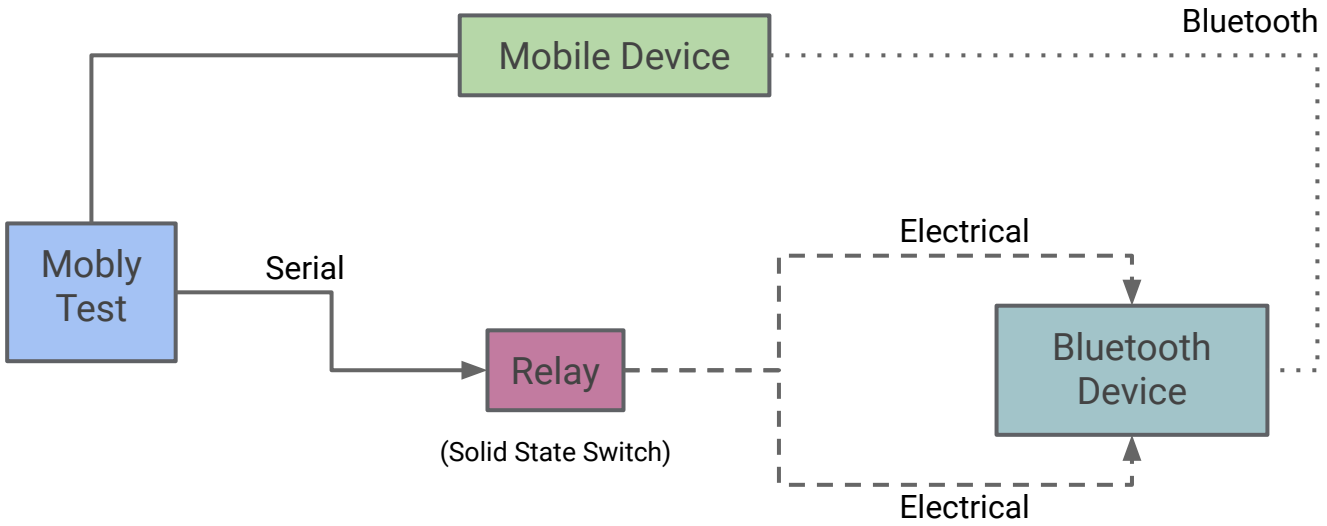
```
class CallTest(base_test.BaseTestClass):
  ...
  def test_simple_call(self):
    with expects.expect_no_raise('Failed to initiate call'):
      self.iphone.makeCall(self.ad.phone_number)
    self.android.acceptCall()

  def on_fail(self):
    self.android.take_bug_report()
    self.iphone.save_syslog()
```

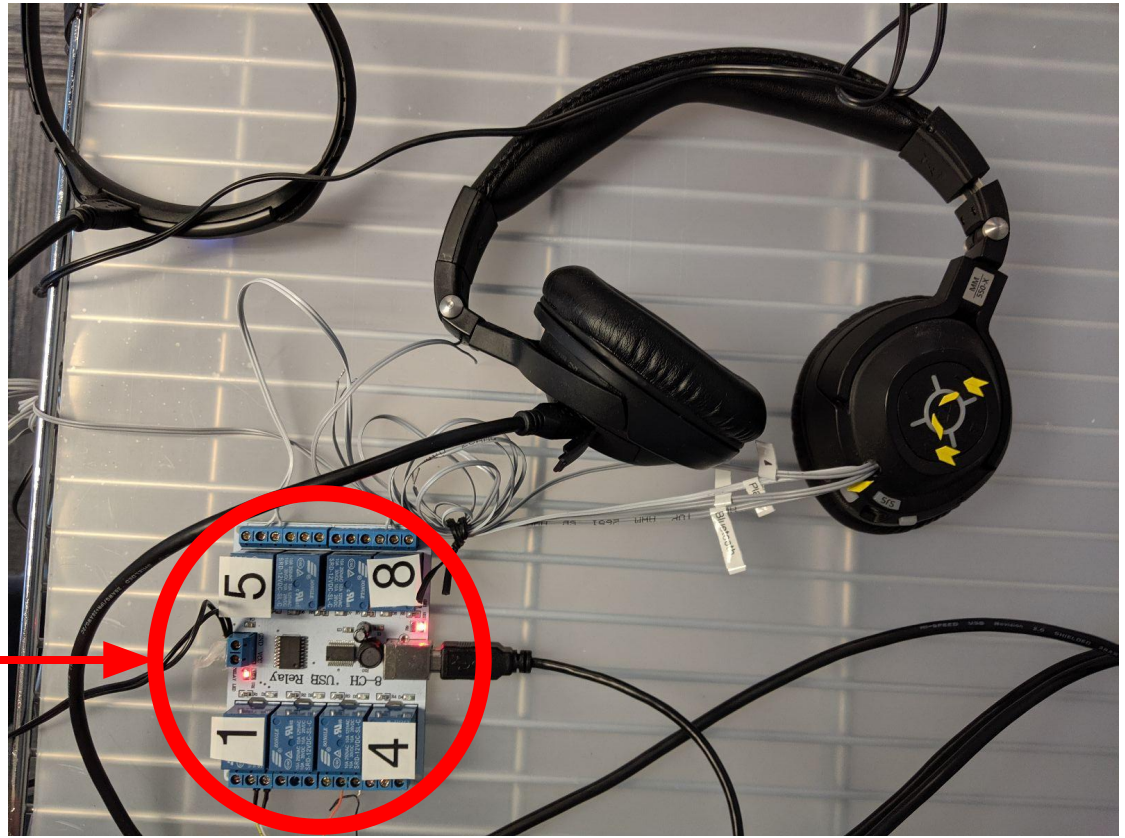
Cross-platform Duo Video Call Demo

Example Use Case:
Pairing Bluetooth Devices

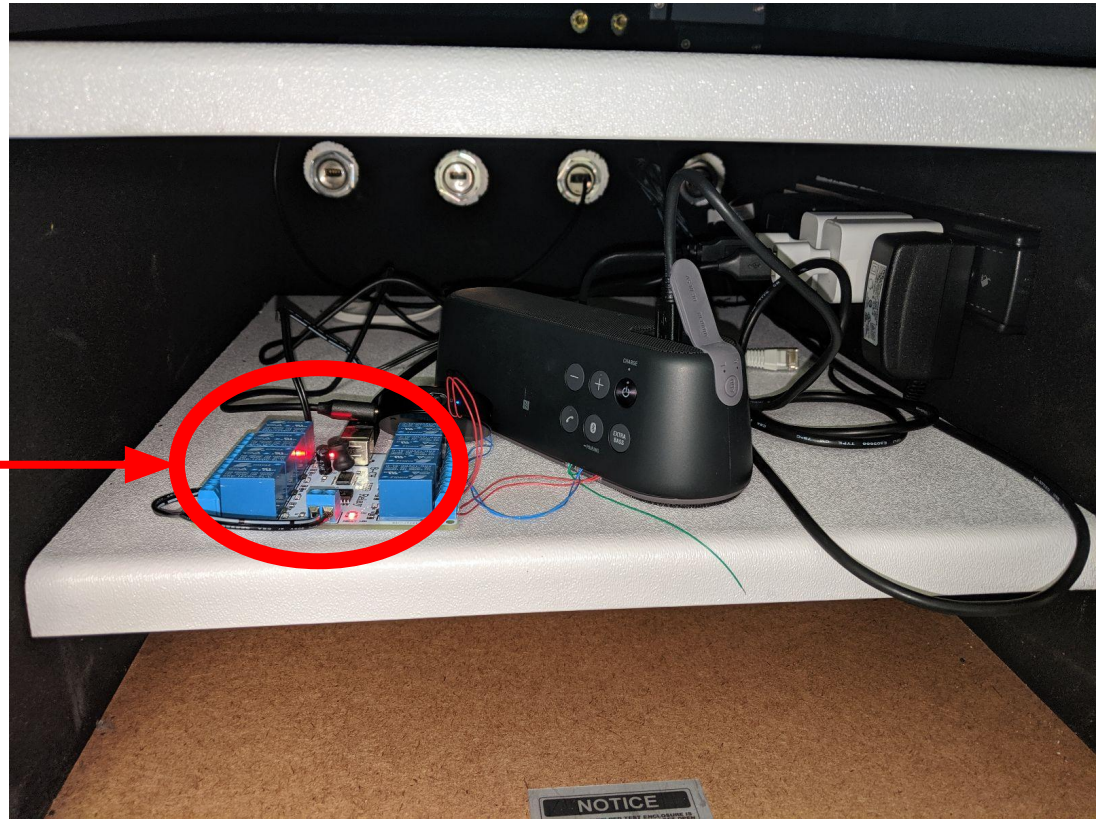
Pairing Bluetooth Devices



Bluetooth Headphones Setup

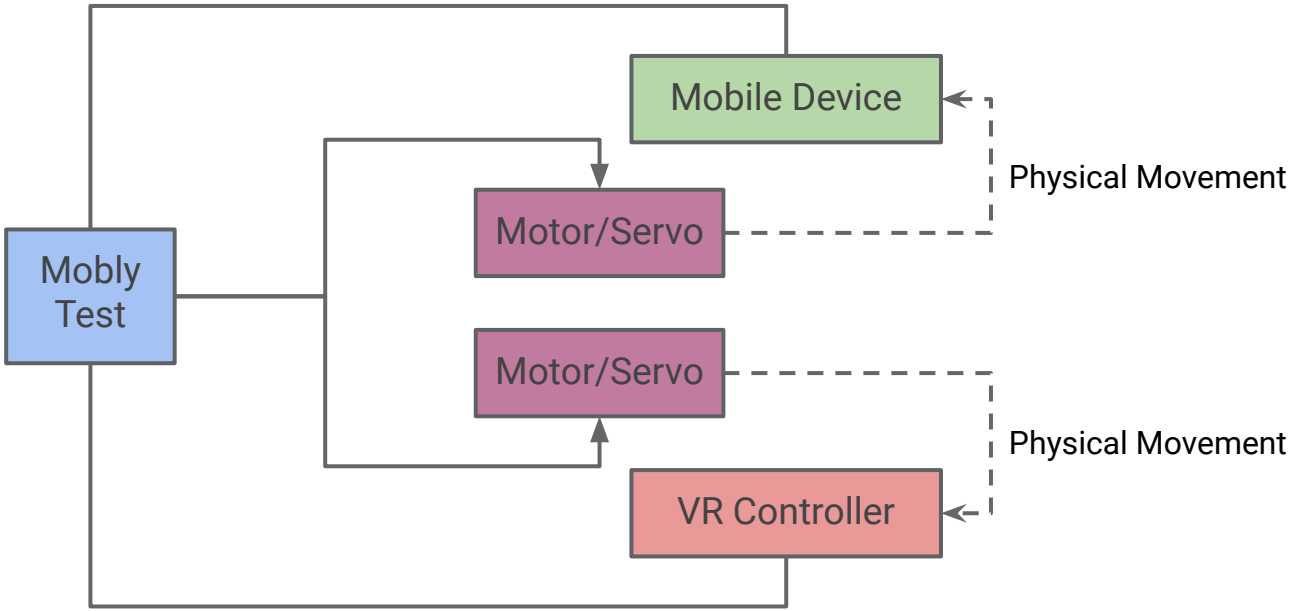


Bluetooth Speaker Setup

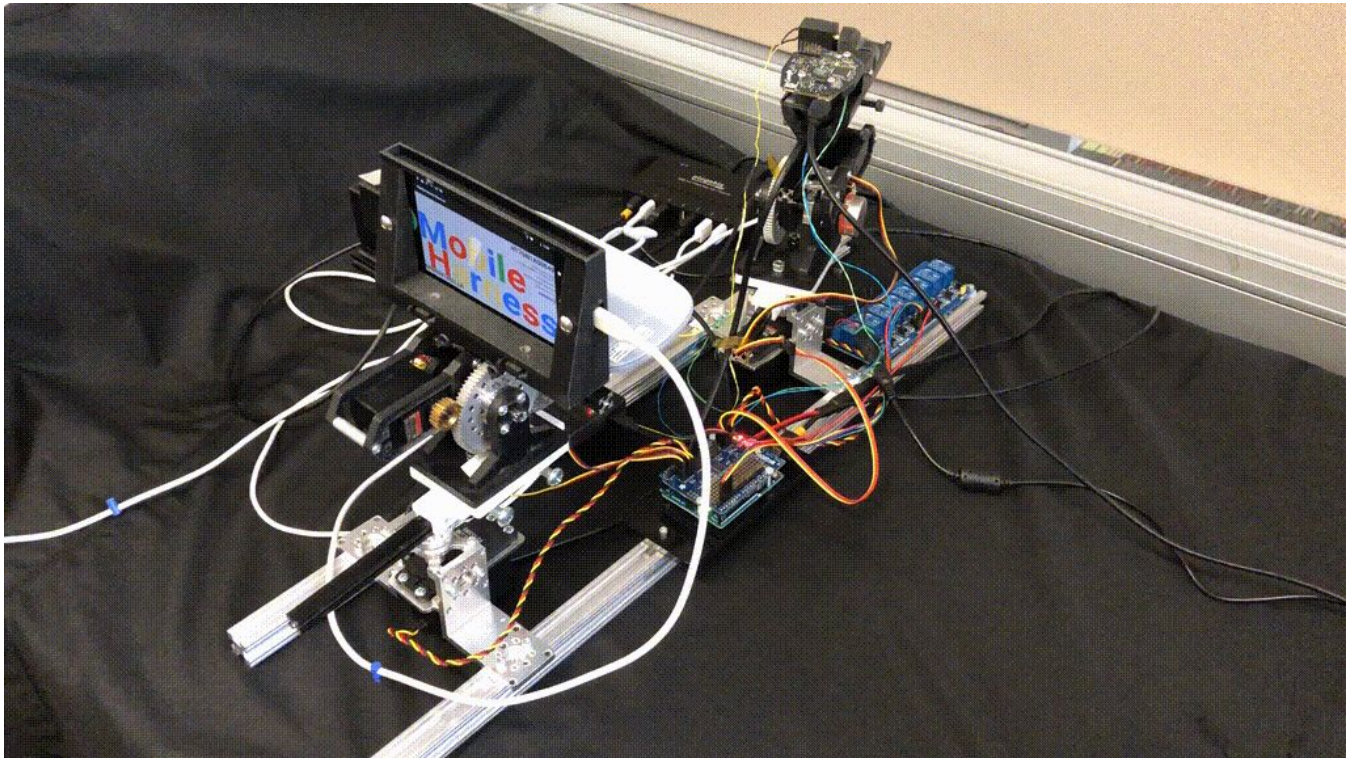


Example Use Case: Simulating Physical Movement in Virtual Reality (VR)

VR Movement Simulation



VR Movement Demo



Widely Adopted in Alphabet

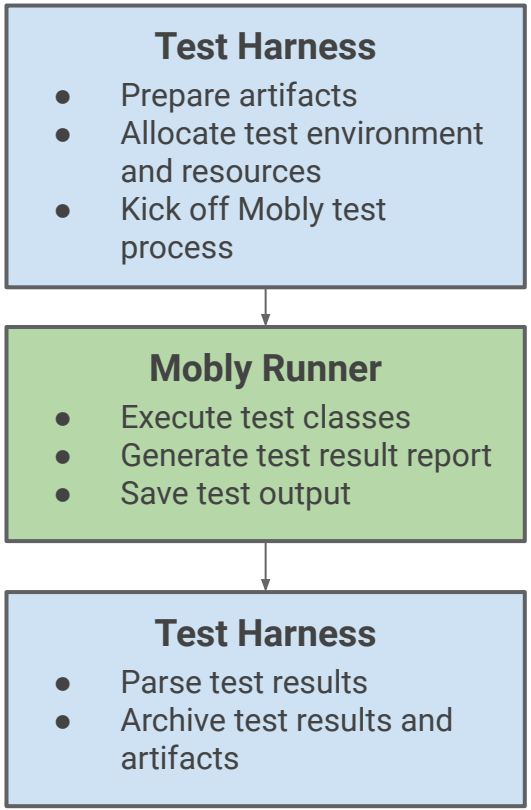
- Telecom
  - Fi
  - Messages
  - Duo
- Google Phone (Pixel)
  - Camera
  - Connectivity
  - Migration Tool
- Android Platforms
  - Auto
  - Things

- VR
- AR/Lens
- Nest
- Loon
- ...

Integration with Test Systems

Using Mobly as part of your test process is easy.

- Main Mobly components are all open sourced on github.
- Mobly framework has APIs designed for creating custom suites.
- Straightforward to integrate with larger scope test systems.



Questions?

References

- Mobly Repo
- Mobly API Doc
- Mobly Snippet Lib Repo
- Mobly Bundled Snippets Repo