An extension of TRIANGLE testbed with model-based testing*

Laura Panizo Almudena Díaz Bruno García
[laurapanizo,almudiaz,bgarcia]@lcc.uma.es

Universidad de Málaga, Andalucía Tech, Dept. de Ciencias de la Computación

Abstract. Traditional testing methods for mobile apps focus on detecting execution errors. However, the evolution of mobile networks towards 5G will require additional support for app developers to ensure also the performance and user-experience. Manual testing in a number of scenarios is not enough to satisfy the expectations of the apps final users. This paper presents the testing framework developed in the TRIANGLE project[1] that integrates a complete mobile network testbed and a model-based testing approach, which is based on model checking, to automatically evaluate the apps performance in different network scenarios.

Keywords: Model-based testing, mobile network testbed, model checking

1 Introduction to Triangle Testing Framework

The TRIANGLE testbed [1] is devoted to the testing and benchmarking of mobile applications and devices. Figure 1 shows an overview of the main functional blocks the testbed architecture.

The testbed provides a high-level access based on a *web portal* whose main purpose is preparing and running tests, and later reviewing the results. It provides an intuitive interface for the definition and execution of the *testing campaigns*, hiding unnecessary complexity. Testing campaigns are based on the execution of the *test cases* specified in the TRIANGLE project for app testing. A test case defines the configuration of the network scenarios, the measurements that have to be collected and the app user flow (sequence of actions) that will be used to activate the feature under test. The network scenarios and the measurement are configured automatically based on the app features, the device or the *high-level scenario* selected by the user. In addition, the the user has to provide the app users flows. All these user inputs are processed and transformed into inputs for the different components of the underlying architecture.

The web portal stores all the campaigns and other user provided data, as well as the results obtained from the testbed, so that test case results are completely traceable to their configuration, and can be repeated if needed.

* This work is funded by the European Union's Horizon 2020 research and innovation programme, grant agreement No 688712 (TRIANGLE project).

[1] https://www.triangle-project.eu/

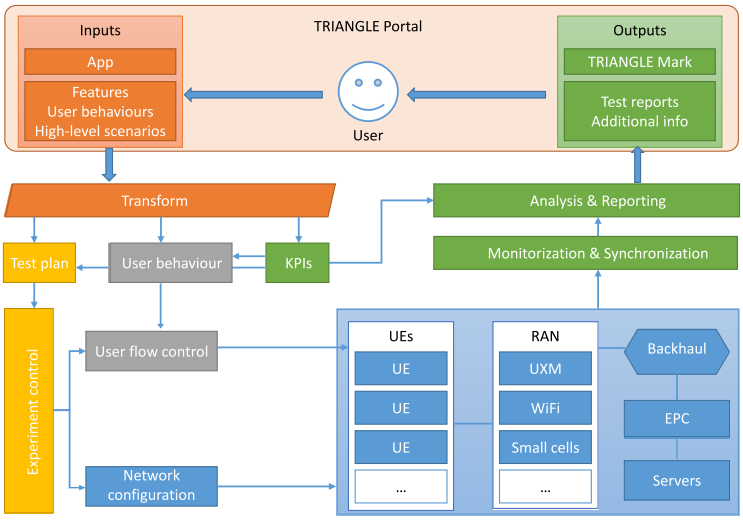

Fig. 1. TRIANGLE testing framework overview

The Experiment Control module coordinates configuration and execution of all network components. The configuration of the different network elements is determined by the high-level scenarios, which abstract similar network configurations that are reproduced during the test case execution. The testbed translates the high-level scenarios into the specific configurations that are applied to each network component.

An integral part of testing apps is automating their execution, i.e. simulating the interactions of an user with the app. Quamotion automation tools[2] provide the means to create sequences of user actions, and then replaying them on a testbed device. In addition, the testbed provides an instrumentation library in order to collect measurements related on the internal performance of the app under test. The source code of the app is instrumented with this library to push measurements points into the logcat, the Android logging system, and correlate these points with radio and power measurements.

The Radio Access Network is provided by a UXM Wireless Test Set from Keysight[3], a mobile network emulator that provides test features such as flexible Inter Cell Interference Coordination (eICIC) schemes or IMS/End-to-End VoLTE communications between multiple devices. The mobile devices, where the applications under test are executed, are physically connected to the testbed. In order to preserve the radio conditions configured at the UXM Wireless Test Set, the radio antenna connection is conducted through cables. In addition, to analyse properly power consumption, the device is powered directly by a power analyser. Finally, the testbed also integrates a commercial EPC from Polaris

[2] http://quamotion.mobi/Mwc
[3] https://www.keysight.com/en/pd-2372474-pn-E7515A/uxm-wireless-test-set

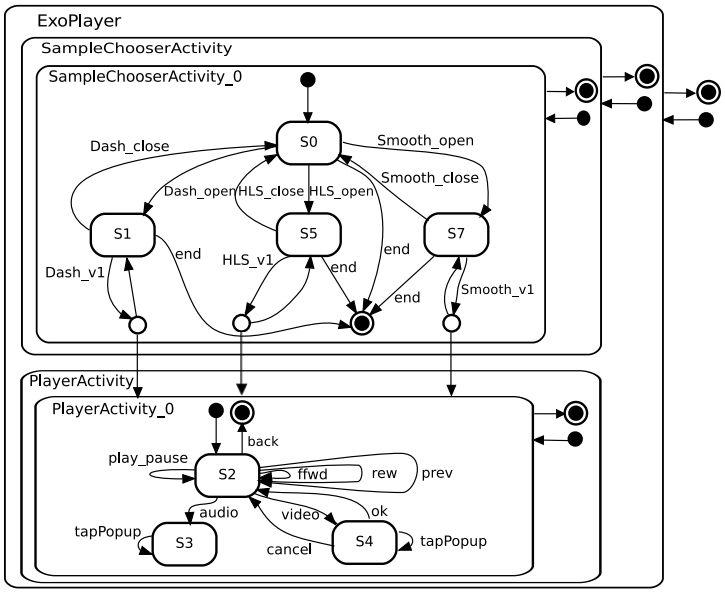

Fig. 2. Exoplayer model with nested state machines

Networks[4], which includes the main elements of a standard core network and local application servers such as streaming or VoIP servers.

The next section presents the model-based testing approach, which is based on model checking, to automatically generate the app user flows.

2 Model-based testing

The TRIANGLE testbed integrates model-based testing techniques to support the automatic generation of app user flows. In [2], we presented a preliminary version of this approach, which is based on the exhaustive exploration of an app model with the model checker Spin [3]. The app model is described with nested state machines [4] that are able to capture the interaction of the user with the app and the intrinsic behaviour of mobile operating systems. Figure 2 shows a model of Exoplayer app, an Android video player that supports different video codecs. The model can be manually generated or automatically extracted from the compiled code of the app. Due to space limits, we do not provide details about the automatic model generation.

The exhaustive exploration of the app model can produce a large number of app user flows, and probably all of them are not suitable for the test cases. In order to generate useful app user flows, the developer defines the set of requirements that the app user flows have to satisfy. The requirements define the states

[4] http://www.polarisnetworks.net/epc-emulators.html

Listing 1.1. Example of requirements in XML format

```
<appUserFlowRequirement xmlns="appuserflowrequirement" name="DemoPlayer_never_1">
  <invariants>
    <event name="play_pause" max="1"/>
  </invariants>
  <sequence>
    <constraint  type="simple">
      <event name="Dash_video_1" min="1" max="1"/>
    </constraint>
    <constraint  type="simple">
      <event name="play_pause" min="1" max="1"/>
    </constraint>
    <constraint  type="simple">
      <state name="end" view="SampleChooserActivity"
        statemachine="SampleChooserActivity_0" visit="true"/>
    </constraint>
  </sequence>
</appUserFlowRequirement>
```

that have to be visited (or not) and the events that can be fired. Listing 1.1 shows an example. This example defines an invariant that forces only one event play_pause in the complete app user flow. In addition, it also specifies a sequence of events and states that have to be visited (relative order). Any other events or states can be fired and visited if the invariant is still satisfied.

The app model and the requirements are uploaded to the web portal. The model is automatically translated into a Promela specification and the requirements are translated into a *never claim*, which is an special Promela process that guides and prunes the exploration of the app model. Then, the Spin model checker verifies the app model against the set of requirements, in such a way that counterexamples represent the app user flows satisfying the requirements. Each counterexample is recorded in the format of Quamotion automation tool, in order to automate the app user flow execution on the device.

3 Conclusions

The TRIANGLE testbed supports app developers to test applications in realistic network scenarios. The integration model-based testing techniques improves the usability and flexibility of the testbed in different ways:

- The testbed automatically produces a pool of app user flows that use the app features in different ways, improving the test coverage.
- App user flows satisfying different requirements are generated with the same app model. If the testbed is extended with new test cases, the app model does not change, and only new requirements have to be defined.
- The app user flow format is transparent for the app developer. Currently, the portal supports app user flows in JSON format, but we plant to migrate to Powershell scripts in the near future due to new functionality integrated in the testbed.

References

1. A. F. Cattoni, G. Corrales-Madueño, M. Dieudonne, P. Merino, A. Díaz-Zayas, A. Salmerón, and ... A. Moore. An end-to-end testing ecosystem for 5g. In *European Conference on Networks and Communications, EuCNC 2016, Athens, Greece, June 27-30, 2016*, pages 307–312, 2016.
2. L. Panizo, A. Salmerón, M. M. Gallardo, and P. Merino. Guided Test Case Generation for Mobile Apps in the TRIANGLE Project: Work in Progress. In *Proc. of the 24th International SPIN Symposium on Model Checking of Software*, pages 192–195. ACM, 2017.
3. G. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, September 2003.
4. A. R. Espada, M. M. Gallardo, A. Salmerón, and P. Merino. Performance Analysis of Spotify® for Android with Model Based Testing. *Mobile Information Systems*, 2017:14, 2017.

Demo: An extension of TRIANGLE testbed with model-based testing

The demo will show the testing and benchmarking process of an app with the TRIANGLE testbed from the point of view of an app developer. Due to time limitations, the instrumentation of the app source code, and the generation of the app model will be carry out before the demo. The results of these tasks will be presented and used as input in the demo.

The TRIANGLE portal is the main tool used by the app developer to test an app. The demo will show how to use the portal to test a sample app and how to interpret the results. The main phases to test an app are:

1. Upload the compiled app (apk file), the app model (previously generated) and the requirements, and select the app features (see Figure 1).
2. The testing framework generates the app user flows using the model-based testing approach. When it finishes, the portal informs about the number of app user flows generated.
3. Define a model-based testing campaign and run it (see Figure 2). The network infrastructure is automatically configured to run multiple test cases, each one with a different app user flow.
4. Exploration of results. The testing framework monitors different magnitudes of the network and the device and reports this information to the developer. In addition, the portal assigns a mark to the app based on the the evaluation of different Key Performance Indicators (KPIs).

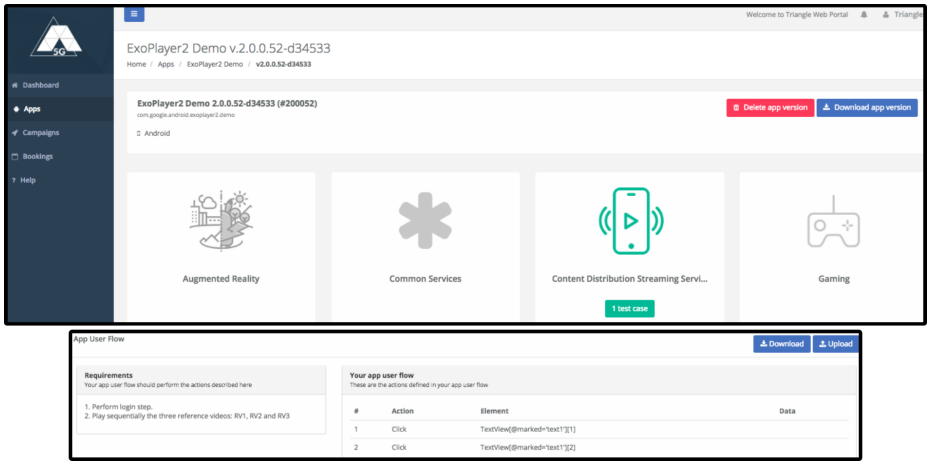

Fig. 1. TRIANGLE portal - App features

Typically, the configuration of the network scenario, the execution of a single test case, and the rendering of results takes around 10 - 15 minutes. For this

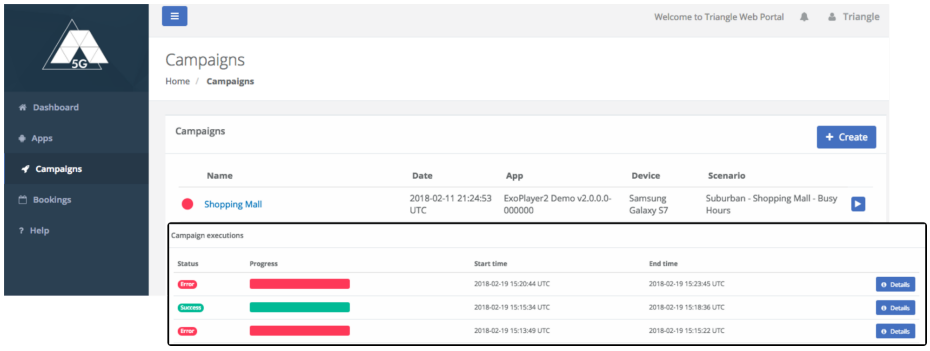

Fig. 2. TRIANGLE portal - Campaign execution

reason, only a test case could be completely executed in the demo. Thus, we will execute some test campaigns in advance to show the benchmarking results.