Hacia la automatización de pruebas funcionales y de rendimiento en Android con algoritmos basados en búsqueda

José Antonio Parejo Maestre[1], Adrián Cantón Fernandez, Carlos Ortiz Prieto, and Manuel Cecilio Pérez Gutierrez

Universidad de Sevilla, Depto. de Lenguajes y Sistemas Informáticos
japarejo@us.es

Abstract. Actualmente existen millones de aplicaciones para smartphone que deben ejecutarse correctamente en entornos software, hardware y de conectividad muy variados y cambiantes. Probar dichas aplicaciones es por tanto un reto importante, para el que ligeras mejoras de la productividad suponen grandes beneficios para desarrolladores y usuarios. Este artículo presenta una primera aproximación para la automatización de pruebas funcionales y de rendimiento en aplicaciones android usando algoritmos basados en búsqueda. La viabilidad de la propuesta se ha validado aplicándola a dos aplicaciones simples. Se han generando casos de pruebas que detectan cierres abruptos en las aplicaciones y maximizan su tiempo de ejecución.

Keywords: testing · pruebas · Android · Search based algorithms.

1 Introducción

La popularización de las comunicaciones móviles y el advenimiento de la era del smartphone ha venido de la mano de una proliferación inusitada de las aplicaciones en estas plataformas. Por ejemplo, la app store de apple contiene más de 2.2 millones de aplicaciones, y la tienda de aplicaciones de google play otros 2.8 millones[1]. Por tanto, probar las aplicaciones en estas plataformas es hoy más importante que nunca, y presenta además retos específicos, como por ejemplo la extrema diversidad de contextos y configuraciones (software y hardware) en las que las aplicaciones deben funcionar, las interacciones e integraciones con otras aplicaciones, y la problemática que supone la necesidad de usar diversos dispositivos físicos o emuladores de los mismos para ejecutar las pruebas. En este artículo presentamos una primera aproximación a la automatización de pruebas funcionales y de rendimiento en aplicaciones android usando algoritmos basados en búsqueda. Concretamente, proponemos técnicas para la generación de pruebas e2e (end-to-end) que usan directamente la interfaz de la aplicación tal y cómo lo haría el usuario. La aproximación ha sido validada usando 2 aplicaciones de prueba y un algoritmo de búsqueda que fue capaz de generar casos de

[1] http://bit.ly/mobileAppsInStores

prueba con un número de acciones fijo, que detectaban cierres abruptos en las aplicaciones, y maximizaban el propio tiempo de ejecución de las pruebas (y por tanto el tiempo de respuesta experimentado por el usuario).

2 Descripción de la propuesta

Consideraremors un caso prueba como una tupla compuesta de un contexto de ejecución definido mediante el conjunto de aplicaciones en ejecución concurrente en el sistema, y una secuencia de acciones sobre la interfaz de usuario de un tamaño determinado. A su vez, una acción es una actividad realizada sobre un control de la interfaz en una de sus vistas. La propuesta propone un proceso que permitirá generar casos de prueba (o conjuntos de estos) que maximizan diversos criterios. A continuación describimos los pasos de dicho proceso.

Identificación del conjunto de contextos de ejecución posibles. En este primer paso se identifican el conjunto de aplicaciones que hay instaladas en el entorno de pruebas, de manera que un subconjunto de las mismas puedan ser lanzadas en paralelo a la ejecución de las acciones sobre la aplicación a probar.

Extracción del grafo de entradas, navegabilidad y acciones de la aplicación. El grafo de entradas, navegabilidad y acciones es una estructura de información que representa los elementos de la interfaz de usuario de la aplicación y sus cambios conforme el usuario realiza acciones sobre ella. Concretamente, el grafo contiene un nodo inicial y un nodo actual que permiten representar el estado actual de las pruebas en ejecución. Para generar el grafo de navegabilidad: i) se arranca la aplicación a probar y se inserta en el grafo el nodo inicial asociado a la primera vista de la aplicación; ii) se explora el conjunto de controles de la vista actual, y se generan las entradas de datos y acciones asociadas a cada control, que se almacenan en una lista de acciones a ejecutar por nodo, incluida la pulsación del botón atrás; iii) posteriormente, se ejecuta cada una de la acciones identificadas, y se comprueba si se ha generado una nueva vista (en base al conjunto de controles accesibles), si es así, se genera un vértice hacia el nuevo nodo del grafo que representa a esta vista, si por el contrario estamos en un nodo ya existente, se genera el arco al nodo correspondiente y se continúan ejecutando su lista de acciones pendientes. El algoritmo termina cuando no queda ninguna acción por ejecutar o la aplicación a probar termina

Generación de pruebas basada en búsqueda. Se genera un caso de prueba mediante un algoritmo que genera un recorrido por el grafo con un número de acciones determinado (por ejemplo, un algoritmo aleatorio), y las ejecuta mientras intenta maximizar alguna función objetivo (como por ejemplo la diversidad de acciones realizadas, o el número de nodos visitados). La secuencia de acciones generada para el caso de prueba que optimiza el criterio puede ser almacenada para su ejecución en diferido.

Ejecución de los casos de prueba generados. Para la ejecución de los casos de pruebas generados se han creado unas clases de soporte, que leen los casos de prueba y ejecutan su secuencia de acciones.

Para la implementación de los pasos del proceso se ha hecho uso de UIAutomator de Google en su versión 2.1.2. Sin embargo, la propuesta es general y podría realizarse sobre implementaciones alternativas con otras herramientas de automatización de pruebas de interfaz de usuario, e incluso para otras plataformas móviles.

3 Aplicabilidad y limitaciones de la propuesta

La aplicación de la propuesta ofrece diversas posibilidades que consideramos de gran interés, son entre otras *la generación de conjuntos pruebas que maximicen la cobertura de nodos del grafo de navegabilidad, y el número de controles sobre los que se ha actuado*. Esta función objetivo permitirá generar pruebas que ejecutan un conjunto de acciones lo más diversas posible en el mayor número de vistas de la aplicación, permitiendo aumentar nuestra confianza en que la aplicación no sufre errrores graves de uso que provoquen cierres inesperados. Otros objetivos de interés serían *la búsqueda de pruebas (secuencias de acciones), de un tamaño determinado que maximicen su tiempo de ejecución*, y la *búsqueda de pruebas que maximicen/minimicen el consumo de batería o memoria*. El uso de esta última función objetivo permitiría identificar puntos calientes en la aplicación en cuanto al consumo de memoria y batería. En última instancias esto permitiría focalizar los esfuerzos de los desarrolladores para mejorar la eficiencia de las aplicaciones, y realizar estas mejoras de manera más eficiente. Finalmente, *La búsqueda de contextos de ejecución* que maximicen el tiempo de ejecución, la memoria o energía consumida o que generen cierres abruptos de la aplicación para un conjunto de tests permitiría evaluar la robustez de la aplicación a las variaciones en su contexto de ejecución.

Actualmente la propuesta tiene varias limitaciones importantes. Para construir el grafo de navegabilidad, se necesita proporcionar generadores de datos de entrada para los distintos controles de las vistas, que la propuesta no contempla, por lo que el grafo generado tiende a ser incompleto en interfaces de usuario complejas. Para paliar este problema se propone usar un fichero de configuración que permita indicar una configuración de generadores de datos. Además, el uso del emulador integrado en Android Studio dificulta la evaluación de rendimiento y la medición del consumo de energía y memoria.Finalmente, sólo se tiene en cuenta como acciones la pulsación de botones y no lanza las aplicaciones especificadas en el contexto de ejecución.

4 Validación

Para validar la viabilidad de la propuesta, se han implementado dos aplicaciones Android. La primera contiene cinco botones, cuya pulsación provoca la ejecución de bucles de espera con distinto número de iteraciones. Además se ha sembrado un error con un botón concreto que lanza de una excepción que provoca el cierre abrupto de la aplicación. Para dicha aplicación se ha implementado un algoritmo de búsqueda aleatoria que es capaz de identificar secuencias de acciones

(pulsaciones de los botones) que provoquen el cierre abrupto de la aplicación, y maximizar el tiempo de ejecución de secuencias de acciones de un tamaño determinado. Esto permite validar que es posible lanzar las pruebas y ejecutarlas así como aplicar un algoritmo de búsqueda para generar casos de prueba en base a una función objetivo. La segunda aplicación contiene dos vistas, una inicial con varios botones, con uno que permite navegar hacia la segunda vista, que contiene un botón que lanza una excepción que provoca el cierre de la aplicación. El algoritmo fue capaz de usar la información del grafo de navegabilidad para identificar una secuencia de acciones que provoca el cierre de la aplicación.

Las aplicaciones y pruebas están disponibiles en http://bit.ly/JISBD2018And. Todas las aplicaciones, algoritmos y pruebas se han ejecutado usando Android Studio 3.0 y el emulador integrado, usando la configuración basada en la API nivel 23 y un Nexus 5 como dispositivo emulado.

5 Trabajo relacionado y futuro

Este trabajo es un primer paso hacia la generación automática de casos de prueba para aplicaciones Android sensibles al contexto de ejecución mediante algoritmos basados en búsqueda. Aunque existen propuestas para la automatización de pruebas y generación de casos y datos de pruebas en Android más sofisticadas [2, 1], la propuesta incorpora aspectos novedosos: la inclusión de un contexto de ejecución de los tests, y funciones objetivo para la evaluación del rendimiento de las aplicaciones en términos del consumo de memoria y energía. En el futuro mejoraremos la generación del grafo de navegabilidad, incorporando más acciones, integrando mecanismos de generación de datos de prueba configurables, y compatibilizándolo con otros formatos de descripción de interfaces de usuario. Posteriormente, aplicaremos diversos algoritmos de búsqueda como GRASP o algoritmos evolutivosm a aplicaciones reales de código abierto.

Agradecimientos

Este trabajo ha sido financiado parcialmente por la Comisión Europea (FEDER) y el gobierno de España bajo el proyecto CICYT BELI (TINN2015-70560-R) y el gobierno de Andalucía mediante el proyecto COPAS (P12-TIC-1867).

References

1. Liu, P., Zhang, X., Pistoia, M., Zheng, Y., Marques, M., Zeng, L.: Automatic text input generation for mobile testing. In: 39th International Conference on Software Engineering. pp. 643–653. ICSE '17, IEEE Press, Piscataway, NJ, USA (2017)
2. Moran, K., Linares-Vásquez, M., Poshyvanyk, D.: Automated gui testing of android apps: From research to practice. In: Proceedings of the 39th International Conference on Software Engineering Companion. pp. 505–506. ICSE-C '17, IEEE Press (2017). https://doi.org/10.1109/ICSE-C.2017.166