Licenciatura en Ciencias de la Computación, Facultad de Ciencias, UNAM.
Programación de dispositivos móviles.
Profesora: Ana Libia Eslava Cervantes.
Ayudante: Manuel Ignacio Castillo López.

Notas de clase

Características de Android como plataforma de desarrollo

Los frameworks de Android

Para desarrollar en Android se utilizan tres frameworks oficiales:

- Android SDK - El Source Development Kit es el framework de desarrollo en Android más utilizado, el más grande (por la cantidad de herramientas y servicios disponibles). Está basado en Java y XML [1].
- Android SDK en Kotlint - Es como el anterior, solo que en lugar de usar Java para escribir la lógica, se usa Kotlint; que es un lenguaje de programación basado en Java. Su sintaxis es similar a la de Python y Javascript [2].
- Android NDK - El Native Development Kit es un framework de desarrollo en Android basado en C++; lenguaje en el que está escrito Android. Provee de ciertas herramientas con las que es posible crear una aplicación completamente funcional, pero no es tan extenso como el SDK, por lo que solo se recomienda utilizar para aplicaciones que requieran de procesamiento en tiempo real, gráficos en 3D, sonido de alta fidelidad y otro tipo de funciones de procesamiento intenso [3].

Una misma aplicación puede desarrollarse usando más de uno de los frameworks; pueden usarse los 3 en el mismo proyecto. Esto permite desarrollar la aplicación modularmente y usar el API más adecuado para los requerimientos de nuestra aplicación. [4]

Existen muchos otros frameworks no oficiales para desarrollar Android, y en general no son compatibles con ninguno de los oficiales. Editores de juegos como Unity3D también pueden exportar aplicaciones de Android.

Hasta hace poco, se acostumbraba utilizar frameworks de desarrollo paralelo. Estos solían usar HTML para indicar la posición de elementos en pantalla y Javascript para la lógica de la aplicación y podían exportar aplicaciones para Android y iOS. El problema de este framework; y muchos otros no oficiales, es que no implementan o no aprovechan las características del sistema en la mejor forma (tampoco para iOS) [5].

Esto usualmente resulta en aplicaciones lentas y con un aspecto visual ajeno al estándar de la plataforma; es decir, resulta en aplicaciones de mala calidad. Las tiendas de aplicaciones están repletas de este tipo de aplicaciones de mala calidad y ahora ni la Apple App Store ni Google Play aceptan aplicaciones basadas en HTML ni de editores en paralelo. Las versiones más recientes de iOS y Android ya incluyen restricciones sobre el uso de vistas Web para evitar el desarrollo de este tipo de aplicaciones y no soportar las aplicaciones ya existentes [6].

Los entornos de ejecución

Una de las decisiones de diseño más eficientes que produjo el equipo de Android Inc, fue el cómo hacer funcionar las aplicaciones en la amplia gama de distintos fabricantes y modelos de celulares en el mercado. Desde antes de los smartphones, ya había un catálogo bastante extenso de arquitecturas de celulares, que se implementan de distinta forma entre fabricantes. Esto también era un impedimento para desarrollar en Symbian, ya que había variantes del sistema operativo por cada una de estas arquitecturas; y a veces hasta entre modelos.

Android Inc. Tomo inspiración de Java ME y decidió que la plataforma de ejecución de Android debería consistir de una máquina virtual. Así, Java se convirtió en el lenguaje de desarrollo de Android, pero no se utilizó la JVM. Android Inc creó su propia implementación de la JVM llamada la *Dalvik Virtual Machine*. La DVM implementa todo el API de Java SE; excepto el paquete java.awt. Además, la DVM tiene optimizaciones sobre funciones críticas para un dispositivo móvil. [1]

Cada aplicación, cada widget y cada servicio, se ejecuta sobre su propia instancia de la DVM. Básicamente, sobre el SO se ejecutan un montón de máquinas virtuales. La gran ventaja de la DVM es que una aplicación solo debe compilarse una vez y cualquier dispositivo Android podrá ejecutarla. [1]

También, desde el inicio Android Inc también creó el NDK, como una alternativa para crear aplicaciones que se puedan ejecutar sin los inconvenientes de una máquina virtual. Esto tenía mucho más sentido cuando Android fue diseñado, pues ahora los celulares se han vuelto tan poderosos en cuanto a capacidad de cómputo, la implementación de intérpretes en tiempo real y tecnología de virtualización, una aplicación escrita con el SDK puede ejecutarse tan rápido como lo habría sido con el NDK.

Actualmente el NDK sigue siendo una poderosa tecnología que nos permite producir aplicaciones de alto rendimiento, pero solo se recomienda cuando sea realmente necesario. La gran desventaja del NDK contra el SDK, es que las aplicaciones creadas con el NDK al no utilizar virtualización, son compiladas para una arquitectura específica de dispositivo y no tiene las ventajas de portabilidad, por lo que desarrollar con el NDK puede ser complicado si queremos expandir nuestros usuarios a todas las arquitecturas de móviles. [7]

Con Android 4.4 Kitkat, la DVM fue hecha obsoleta en favor de ART (Android Runtime). ART es mucho más eficiente que la DVM. Una de las mejoras que incluye es que cuando se instala una aplicación, es compilada en código objetivo para el sistema anfitrión, de manera que se pueda ejecutar directamente. En consecuencia, las aplicaciones ocupan más espacio en almacenamiento. A partir de Android 5 Lolipop, la DVM fue removida por completo de Android. [8]

Cada aplicación juega en su propia cajas de arena

Mencionamos anteriormente que cada aplicación se ejecuta en su propia máquina virtual. Y aunque la DVM ya no usa, aún el espacio de cada aplicación es privado por completo. Sus archivos, el caché, sus bases de datos, y todos sus componentes son invisibles;

inexistentes para otras aplicaciones. Es posible definir algunos componentes como público, como algunas tablas de la base de datos, y definir acciones para poder responder a señales emitidas a todo el sistema. Es posible hacer interactuar dos aplicaciones sin tener que definir memorias compartidas o túneles de datos. [1]

Filosofías de uso e implementación

Como todas las plataformas, Android tiene ventajas y desventajas frente a distintos casos de uso. Empecemos por analizar algunos de los contras:

- Poca memoria. Si bien hay dispositivos de Android con más de 2Gb de memoria, el uso de memoria por parte de las aplicaciones debe ser moderado. Una aplicación que consume demasiada memoria, será lenta y puede garantizar también el desempeño del sistema si se pone a ejecutar en segundo plano. Además, dependiendo nuestro público objetivo, la mayoría de las veces no podremos descartar los dispositivos con menos memoria.

  Una buena práctica respecto al manejo de memoria es tratar de repartirlo. Tratar de delegar los gráficos a OpenGL por ejemplo. Esto solo es necesario en aplicaciones que requieran de un gran uso de memoria, para la mayoría es suficiente con el SDK estándar y las herramientas que proporciona optimizar memoria y recursos; como RecyclerView o AsyncTask.

- Duración de la batería. Muchos de los dispositivos Android son smartphones; y a menos que desarrollemos para dispositivos que siempre están conectados a la corriente (como Android TV), tenemos que evitar que nuestra aplicación sea responsable por el rápido consumo de la batería; al menos cuando esté en segundo plano.

  Toda función de cómputo intensivo, el uso de sensores, hilos en ejecución y dispositivos como el vibrador y la pantalla, hacen que la batería se agote pronto. Esta regla no aplica del todo para una aplicación en ejecución; pues caer en cualquiera de los casos anteriores significa que la aplicación necesita de esas características (a menos que se presenten errores de diseño o programación).

  Además que en tiempo de ejecución tenemos que tratar de minimizar el consumo de batería, es importante que cuando la aplicación entre en segundo plano se detengan los hilos adicionales, se regrese el brillo de la pantalla a su configuración original, se detenga la escucha de sensores; etc. Si no lo hacemos, nuestra aplicación podría agotar rápidamente la batería de los dispositivos y puesto que se puede saber el consumo de batería de cada aplicación que se ha ejecutado desde la última carga, será rechazada por sus usuarios y dificultará la obtención de nuevos usuarios.

- Interrupciones. Durante el uso de una aplicación el usuario es interrumpido por diversos factores; por ejemplo: recibir una llamada. Las notificaciones no necesariamente lo interrumpen por completo, pero es necesario tomar en cuenta que en cualquier momento puede mostrarse una notificación y podría ocultar algún botón, imagen o texto en la pantalla; y aunque sea por un breve momento, esto

puede desconcertar y frustrar a los usuarios, por lo que es importante tratar de no colocar elementos importantes en las áreas donde se suelen mostrar notificaciones y también diseñar nuestras notificaciones con el contenido más breve y concreto posible. También se acostumbra usar un relleno vacío en los primeros elementos de las vistas.

No solo el usuario puede ser interrumpido por el sistema en sí, los usuarios de dispositivos móviles suelen usarlos por cortos periodos de tiempo, muchos de los cuales pueden ser interrumpidos por alguna persona o el inicio de un evento. Las aplicaciones deberían tomar esto en cuenta y ayudar al usuario a retomar lo que estaba haciendo antes de distraerse. Muchas veces esto se logra simplemente con guardar el estado de la pantalla.

- Capacidad de cómputo. Aunque existen muchos smartphones bastante poderosos, la mayoría de dispositivos Android activos entran en la categoría de “gama media”, mientras que los de “gama baja” representan un número para nada despreciable. Así que al desarrollar no debemos confiarnos en que los usuarios tendrán un dispositivo con características sobresalientes, es mejor hacer un diseño e implementación adecuado para que nuestras aplicaciones sean lo más eficientes posible.

Por otro lado Android destaca en los siguientes casos de uso:

- Sensores. Android soporta una variedad de sensores, desde acelerómetros y giroscopios, hasta barómetros y sensores de humedad. Las interacciones del dispositivo con el usuario pueden extenderse con el uso de sensores.

- Portabilidad. La mayoría de los dispositivos Android son portátiles y podemos hacer uso de las ubicaciones físicas como parte de nuestra aplicación.

- Conectividad. Android soporta varios estándares de conectividad; desde GSM y 4G, hasta NFC y Bluetooth.

- API extenso.

- Capacidad de integración con productos de Google como Authentification, FireBase, Youtube, Pay, AdMob; entre otros.

- Por defecto, Android usa SQLite, pero podemos incluir un manejador de base distinto, como MongoDB. Además, es posible conservar la información anterior de una versión previa de la base de datos; incluso cuando cambiamos de manejador.

- Soporte de OpenGL y OpenSL.

Al construir una aplicación para Android es importante tomar en cuenta la plataforma, su interfaz, la lógica y su presentación se deben concebir primero como un proyecto de Android, tomando en cuenta patrones y estándares como Material Design.

De debemos tratar un proyecto Android de la misma forma con la que se trata un proyecto en Java o Linux. Hay que tomar en cuenta el tipo de usuarios para los que estamos desarrollando, el tipo de interacciones que esperan y considerar las limitaciones de los dispositivos que usan; para que nuestra aplicación sea lo más eficiente y robusta posible en esas plataformas; y en otras de ser posible.

Referencias

1. Meier, R., (2012). *Professional Android 4 Application Development*. Indiana, Wrox.
2. Google, (2019). “Kotlin y Android” en *Android Developers*.[En línea]. California, disponible en: https://developer.android.com/kotlin/ [Accesado el día 1 de febrero del 2019].
3. Google, (2019). “Como comenzar a usar el NDK” en *Android Developers*. [En línea]. California, disponible en: https://developer.android.com/ndk/guides/ [Accesado el día 1 de febrero del 2019].
4. Google, (2019). “Funciones de Android Studio” en *Android Developers*. [En línea]. California, disponible en: https://developer.android.com/studio/features [Accesado el día 1 de febrero del 2019].
5. Stackoverflow, (2019). “Does app store or / and Play Store allow apps that have a WebView only to my site?” en *Questions*. [En línea]. Nueva York, disponible en: https://stackoverflow.com/a/31194912/2278847 [Accesado el día 1 de febrero del 2019].
6. Baxter-Reynolds, M., (2013). “Here’s why HTML-based apps don’t work” en *ZDNet*. [En línea], San Francisco, disponible en: https://www.zdnet.com/article/heres-why-html-based-apps-dont-work/ [Accesado el día 1 de febrero del 2019].
7. Google, (2019). “Conjunto de herramientas independiente” en *Android Developers*. [En línea], California, disponible en: https://www.zdnet.com/article/heres-why-html-based-apps-dont-work/ [Accesado el día 1 de febrero del 2019].
8. Stackoverflow, (2015). “What is difference between DVM and ART? Why DVM has been officially replaced with ART in Lollipop?” en *Questions*. [En línea]. Nueva York, disponible en: https://stackoverflow.com/a/31958806/2278847 [Accesado el día 1 de febrero del 2019].