Introducción

Android es un sistema operativo basado en Linux, diseñado para ser usado en dispositivos móviles como smartphones o tablets.

En la actualidad, es uno de los sistemas operativos para dispositivos móviles más utilizados.
Android cuenta con un framework de desarrollo completo y de fácil uso.

Este documento describe el diseño e implementación de una aplicación android, desarrollada como trabajo final para la materia “Introducción a la Programación de Dispositivos Móviles”.

La aplicación desarrollada, cumple con los siguientes requerimientos:

El trabajo consiste en realizar un reproductor de audio para Andriod. El mismo deberá usar los ContentProviders de la información multimedia para permitir seleccionar el artista y disco que se desea escuchar. También debería permitir seleccionar un disco, listando todos los discos disponibles. Mínimamente el reproductor debería tener las siguientes características:

- *La música no debería detenerse si la aplicación va a segundo plano (Foreground service).*
- *Mientras se encuentre reproduciendo, debería mostrarse un icono en la barra de notificaciones*
- *Proveer un App Widget para controlar la reproducción desde el home screen.*
- *Cuando se encuentre en la pantalla de reproducción, la aplicación debería mostrar información de la banda obtenida de Wikipedia u otra fuente.*

La sección de desarrollo de este documento, describe cómo se implementó cada uno de los requerimientos descritos.

Desarrollo

A los fines de cumplir con los requerimientos enunciados, se creó la aplicación *AudioPlayer.* La misma está compuesta de un conjunto de clases encargadas de implementar estos requerimientos. Las clases implementadas, junto con las relaciones entre las ellas, pueden observarse en el diagrama de clases de la figura 1.

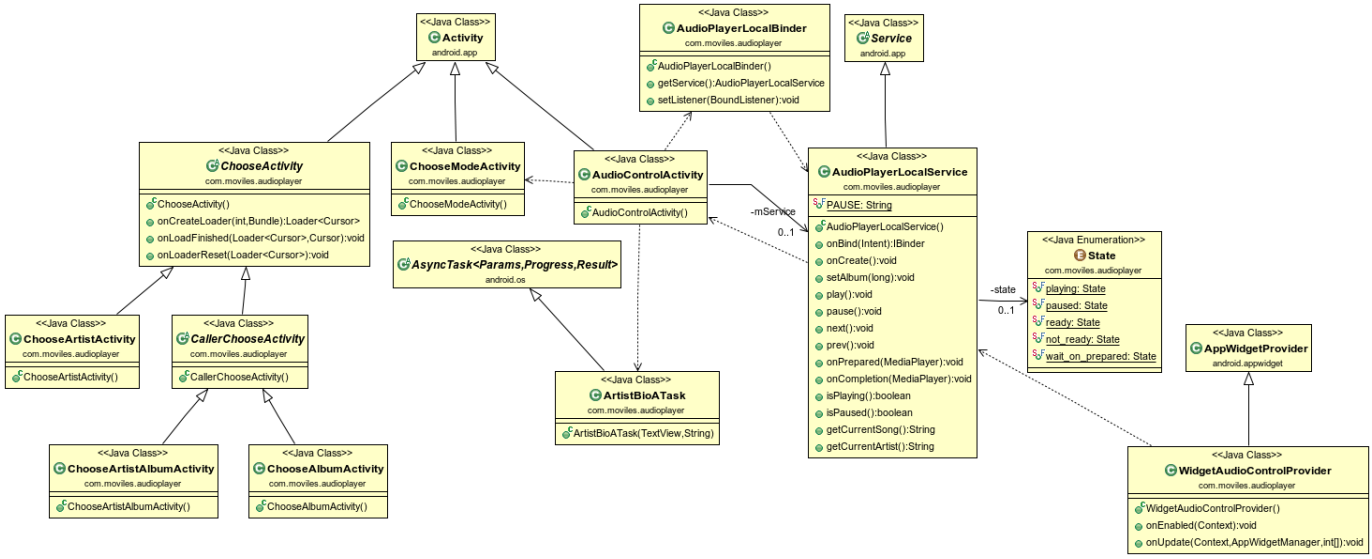

Figura 1 - Diagrama de clases de la aplicación AudioPlayer

Reproducción de audio

Para proveer a la aplicación con la capacidad de reproducir audio, se implementó la clase *AudioPlayerLocalService*. Esta extiende la clase *Service* provista por el framework Android.

Modo de ejecución

Este servicio es iniciado por la clase *AudiocontrolActivity* al comenzar la ejecución de la aplicación. Luego de ser iniciado, el mismo pasa a ejecutarse como un “foreground service” mediante una llama a la función *startForeground* dentro del metodo *onCreate*. El método onCreate, es llamado por única vez cuando el servicio es iniciado mediante la función *startService*.

Interfaz de control

El servicio cuenta con una instancia de la clase *MediaPlayer* provista por el framework Android que es utilizada para reproducir audio.

Los clientes del servicio, pueden controlar la reproducción de audio mediante la interfaz provista por el mismo. Internamente, el servicio almacena el estado actual de la reproducción en una variable. El estado interno, es utilizado para determinar la validez de las llamadas a los métodos de su interfaz. Por ejemplo, si se llama al método *pause* del servicio, el mismo verifica que se encuentre en el estado *playing* antes de llamar al método *pause()* de su instancia de *MediaPlayer*.

La interfaz cuenta con los siguientes métodos:

setAlbum(long albumId): Mediante este método, se le indica al servicio el álbum (y por lo tanto la lista de canciones) que debe reproducir. Internamente, el servicio cuenta con una instancia de la clase *Cursor* que utiliza para acceder a las canciones mediante un *ContentProvider* y a su vez mantener un puntero a la canción que se esta ejecutando en determinado momento.

play(): Comienza la reproducción de la canción actual. Las llamadas a este método son válidas luego de una llamada a *setAlbum* o luego de haber pausado la canción que se estaba reproduciendo.

pause(): Pausa la reproducción de la canción actual. Las llamadas a este método son válidas si se estaba reproduciendo una canción.

next(): Avanza hacia la siguiente canción del álbum. Internamente, avanza el cursor hacia la siguiente fila y le indica a su instancia de *MediaPlayer* que cargue la canción correspondiente a esa fila. Las llamadas a este método son válidas a menos que el cursor esté apuntando a la última canción del álbum.

prev(): Implementado de forma similar al método *next*, retrocede hacia la canción previa a la actual. Las llamadas a este método son válidas si el cursor no esta posicionado en la primer canción del álbum.

isPlaying(): Retorna verdadero si la instancia de *MediaPlayer* está reproduciendo una canción.

isPaused(): Retorna verdadero si la instancia de *MediaPlayer* está en pausa.

getCurrentSong(): Retorna el nombre de la canción en la que esta posicionado el cursor.

getCurrentArtist(): Retorna el nombre del compositor de la canción en la que esta posicionado el cursor.

Notificaciones

Además de proveer una interfaz de control, el servicio *AudioPlayerLocalService*, es el encargado de comunicarle al usuario el estado actual de la reproducción. Esto incluye:

- Notificar al usuario que el servicio está en ejecución.
- Notificar al usuario cuando se pasa de una canción a otra.
- Mostrar el nombre y el artista de la canción actual tanto en la barra de notificaciones como en la actividad *AudioControlActivity*.

Ya que el servicio *AudioPlayerLocalService* se ejecuta en modo foreground, debe proveer una notificación para que el usuario sepa que que el servicio se esta ejecutando. Esta notificación se pasa como parámetro al método *startForeground*. La misma, provee un *PendingIntent* de modo que cuando el usuario seleccione la notificación presente en la barra de notificaciones, se muestre la actividad principal de la aplicación (en este caso *AudioControlActivity*).

Para notificar al usuario del estado actual de la reproducción, se implementó el método *updateNotification* que es llamado cada vez que se comienza a reproducir una canción a través del método *play()*. Este método verifica si la información de reproducción cambio desde la última llamada a *updateNoification*. En caso de que sea necesario emitir una notificación nueva, realiza dos acciones:

En primer lugar, crea una nueva notificación con el nombre de la canción actual y el nombre del artista de dicha canción. Esta notificación la envía a través del *NotificationManager* del sistema. De esta forma, el usuario verá en la barra de notificaciones una nueva notificación con la información mencionada.

En segundo lugar, le indica a la actividad *AudioControlActivity* la canción y el artista de la misma, a través de una llamada a los métodos *setCurrentSong* y *setCurrentArtist* de la interfaz *BoundListener*. De este modo, la actividad *AudioControlActivity* puede actualizar su vista para reflejar el nuevo estado de la reproducción.

Comunicación con App Widget

Por último, el servicio *AudioPlayerLocalService* provee una interfaz de control al *App Widget* implementado en la clase *WidgetAudioControlProvider*.

Esta interfaz se implementa en *AudioPlayerLocalService* simplemente mediante una instancia de la clase *BrodcastReceiver*. Esta, implementa el método *onReceive* que recibe un *Intent* como parámetro, el cuale determina qué acción se debe realizar, mediante el método *getAction* del *Intent*. Las posibles acciones son: *PLAY, PAUSE, NEXT* y *PREV.* Estas se traducen en llamadas a los métodos *play()*, *pause()*, *next()* y *prev()*.

Control de la reproducción

El control de la reproducción puede realizarse mediante dos interfaces, la primera es la que provee la aplicación al ingresar a través de la actividad implementada en la clase *AudioControlActivity*, la segunda es a través del widget implementado en la clase *WidgetAudioControlProvider*. Ambas interactuan con el service *AudioPlayerLocalService.*

Control a través de *AudioControlActivity*

Al iniciar la aplicación se lanza la actividad *AudioControlActivity*. La misma, en el método *onStart*, crea un intent para iniciar el servicio *AudioPlayerLocalService*. Este es iniciado a través del método *startService.* Una vez iniciado, la actividad llamadora se asocia al servicio utilizando el método *bindService*.

Ya que el usuario puede salir de la actividad *AudioControlActivity* para luego iniciarla en algún otro momento, es necesario que que la asociación entre la actividad y el servicio se realice en el método *onStart* de la clase *Activity*.

A su vez la actividad debe desasociarse del servicio en la ejecución de su método *onStop*.

La figura 2 ilustra el ciclo de vida de una actividad. En la misma se puede observar los momentos en los que se ejecutan los métodos *onStart* y *onStop.*

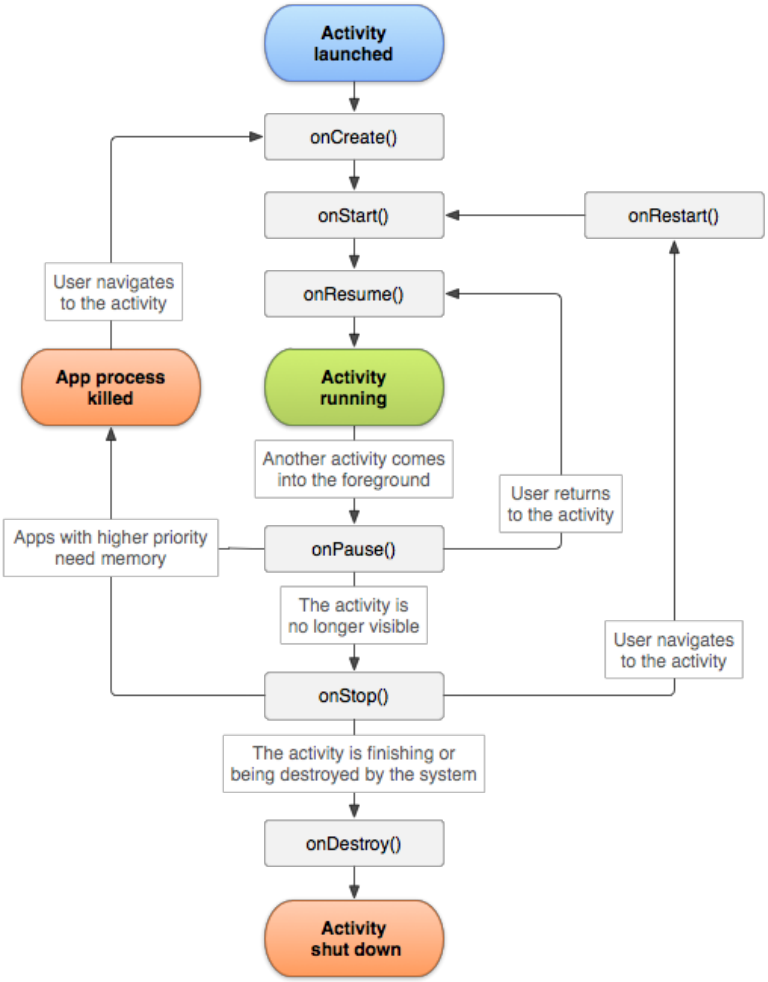

Figura 2 - Ciclo de vida de una actividad

Para hacer posible la comunicación entre la actividad y el servicio, es necesario implementar el *ServiceConnection* del lado del cliente (*AudioControlActivity*), e implementar el método *onBind* en el servicio (*AudioPlayerLocalService*). El método *onBind* devuelve una instancia de la clase *AudioPlayerLocalBinder* cuando una actividad cliente se asocia a este servicio. A través de esta interfaz, la actividad puede obtener la instancia del servicio que está en ejecución.

Control a través de *WidgetAudioControlProvider*

El widget implementa cuatro intents:

- **prevIntent:** Crea el intent indicando el string de acción con la constante *AudioPlayerLocalService.PREVIOUS*
- **playIntent:** Crea el intent indicando el string de acción con la constante *AudioPlayerLocalService.PLAY*
- **pauseIntent:** Crea el intent indicando el string de acción con la constante *AudioPlayerLocalService.PAUSE*
- **nextIntent:** Crea el intent indicando el string de acción con la constante *AudioPlayerLocalService.NEXT*

Luego para cada uno de estos *intents* se crea un *PendingIntent* (*prevPendingIntent, playPendingIntent, pausePendingIntent* y *nextPendingIntent* respectivamente) que se utilizan para realizar la difusión (broadcast) de la acción. Como se mencionó anteriormente (Comunicación con App Widget), el service se registra a estas acciones a través de su implementación del *BrodcastReceiver*. Por último, se vinculan los las instancias de *PendingIntent* a cada uno de los botones del widget.

Selección de álbum

Entre los requerimientos se indica que es necesario permitir al usuario seleccionar un álbum a partir de una lista de álbumes de un artista en particular o desde una lista de albums directamente.
Para brindar esta funcionalidad, se implementaron cuatro actividades:
ChooseModeActivity: Esta actividad muestra dos botones. El primero de ellos inicia la actividad *ChooseAlbumActivity* para permitirle al usuario seleccionar un álbum de la lista de álbumes. El segundo, inicia la actividad *ChooseArtistActivity* la cual permite al usuario seleccionar un artista para luego mostrar la lista de álbumes de este artista.

ChooseAlbumActivity: Esta actividad le muestra al usuario la lista de todos los álbumes que contiene el dispositivo de almacenamiento.

ChooseArtistActivity: Esta actividad le permite al usuario seleccionar uno de los artistas existentes en el dispositivo de almacenamiento. Cuando se selecciona uno de los artistas, esta actividad inicia la actividad *ChooseArtistAlbumActivity*.

ChooseArtistAlbumActivity: Esta actividad le permite al usuario seleccionar un álbum de la lista de álbumes correspondientes al artista previamente seleccionado.

Invocaciones entre actividades
El diagrama de secuencia de la figura 3 muestra el orden de las invocaciones entre actividades.
Para iniciar una actividad, la actividad llamadora debe crear un *Intent* en el cual se indica el nombre de la clase de la actividad que se quiere iniciar. Una vez creado el *Intent*, se llama al método *startActivityForResult* el cual inicia la actividad. Cuando la actividad llamada finaliza, el framework invoca al método *onActivityResult* de la actividad llamadora. Este método recibe como parámetro un *Intent* creado por la actividad llamada. De esta forma, la actividad llamada puede retornar datos dentro de un *Intent,* a la actividad llamadora.
Dentro de las llamadas del diagrama de secuencia de la figura 3, cuando la actividad *ChooseArtistActivity* invoca a la actividad *ChooseArtistAlbumActivity* mediante el método *startActivityForResult(Intent i)*, el *Intent* pasado como parámetro contiene el id del artista seleccionado. De esta forma la actividad *ChooseArtistAlbumActivity* puede mostrar sólo los álbumes del artista seleccionado en la actividad *ChooseArtistActivity*.

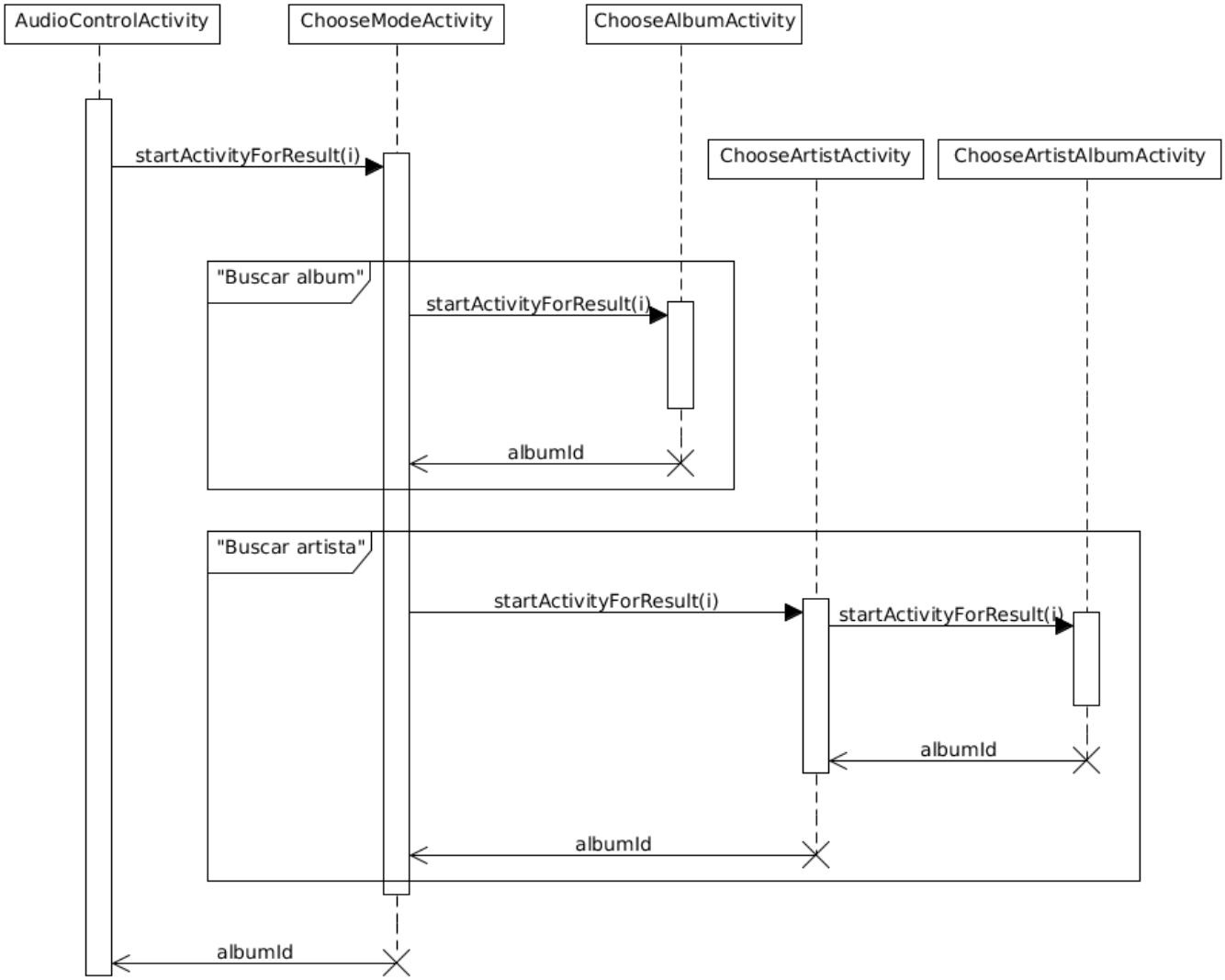

Figura 3 - Diagrama de secuencia para la selección de álbum

Obtención de artistas y álbumes

Las actividades *ChooseAlbumActivity, ChooseArtistActivity* y *ChooseArtistAlbumActivity* deben listar álbumes, artistas y álbumes de un artista respectivamente. Para acceder a los álbumes y artistas contenidos en el dispositivo de almacenamiento externo, se hizo uso de la clase *MediaStore* provista por Android, en conjunto con la clase *Cursor*.

Ya que las actividades muestran una interfaz gráfica, es necesario que las consultas a la base de datos, provista por *MediaStore,* se hagan de manera asincrónica. De esta forma, el hilo de ejecución de la interfaz gráfica no debe detenerse en espera de que se complete la consulta.

A los fines de poder realizar estas consultas de forma asincrónica, las clases previamente mencionadas implementan la interfaz *LoaderCallbacks<Cursor>* provista por el framework.

Esta interfaz cuenta con los métodos *onCreateLoader* y *onLoadFinished*.

onCreateLoader: Este método es el encargado de crear una instancia de la clase *CursorLoader* que es utilizada para indicar los parámetros a utilizar en la consulta a la base de datos. La llamada a este método es responsabilidad del framework.

onLoadFinished: Este método es llamado una vez que se terminó de realizar la consulta a la base de datos. Como parámetro recibe una instancia de *Cursor* la cual apunta a la primer fila de las correspondientes a la consulta. En este punto, las actividades asocian este cursor con la vista del tipo *ListView* que contienen.

Información del artista

La actividad *AudioControlActivity* es la encargada de mostrar información del autor de la canción que se está reproduciendo. Para obtener esta información se utilizó la API provista por el servicio web "The Echonest"[1]. Este servicio provee información sobre canciones y artistas, recuperada de varias fuentes. A su vez, cuenta con una librería implementada en Java para facilitar las consultas.
Las consultas realizadas a la API involucran un request HTTP y por lo tanto pueden tomar un tiempo considerable. Ya que la actividad *AudioControlActivity* ejecuta en su hilo la interfaz gráfica que se le muestra al usuario, se optó por implementar las consultas de forma asincrónica.
La clase *ArtistBioATask* es la encargada de realizar estas consultas de forma asincrónica. La misma extiende la clase *AsyncTask* provista por el framework. Esta, cuenta con dos métodos que se deben implementar: *doInBackground* y *onPostExecut*e.
Cuando la actividad *AudioControlActivity* recibe la notificación por parte del servicio *AudioPlayerLocalService* indicando que el artista de la canción actual es diferente al de la canción anterior, la actividad crea una instancia de *ArtistBioATask* y llama al método *execute* de la misma. De este modo, la tarea se ejecuta el método *doInBackground* en su propio hilo. Al terminar la ejecución, actualiza la vista de *AudioControlActivity* encargada de mostrar la información del autor de la canción actual.

[1]: The Echonest http://www.echonest.com/