Componentes básicos de Android

Curso *Aplicación de Android en el desarrollo de sistemas de Inteligencia Ambiental y Robótica*
Cloud Incubator Hub

Nieves Pavón Pulido

23 de enero de 2013

Componentes básicos de Android

- Descripción breve de los componentes básicos de Android
- Ejemplo de APLICACIÓN ANDROID simple
  - Selección apropiada de componentes
  - Análisis de los componentes Android utilizados: Activity, Service y BR
  - Fichero AndroidManifest.xml

Componentes básicos de Android

Descripción breve de los componentes básicos de Android

Activity: Una actividad o **activity** es el componente básico que permite crear componentes que interaccionan con el usuario.
Cada actividad tiene asociada una vista o **view**. La vista constituye el conjunto de elementos gráficos que se presentan al usuario para que interaccione con el sistema. Por ejemplo, cajas de edición, etiquetas o botones, entre otros.
Para crear actividades personalizadas es necesario definir una clase que herede de la clase base *Activity*.

Intent: Una intención o **intent** es un componente básico de Android que permite el envío de mensajes entre componentes. El paso de mensajes a través de *intents* puede considerarse una facilidad para asociar (*binding*) componentes de forma tardía y en tiempo de ejecución en la misma aplicación o entre diferentes aplicaciones.

Componentes básicos de Android

Descripción breve de los componentes básicos de Android

Service: Un servicio o **service** es un componente básico de Android que permite llevar a cabo operaciones en segundo plano o **background** y, por tanto, no proporcionan al usuario una interfaz de usuario.

Un componente puede asociarse o enlazarse (*bind*) a un **service** para interaccionar con él e incluso llevar a cabo tareas de comunicación entre procesos.

Los **services** pueden tomar dos formas esencialmente:

- **Started**: Una vez iniciado el servicio, se ejecuta de forma indefinida en segundo plano, incluso si se destruye el componente que lo inició.
- **Bound**: En este caso, el servicio ofrece una interfaz de tipo cliente-servidor a aquellos componentes que se asocian o enlazan (*bind*) a dicho servicio. Los componentes asociados o enlazados (*bound*) al servicio pueden enviar peticiones, conseguir resultados y respuestas o, incluso, llevar a cabo tareas de comunicación con otros procesos. En este caso, el servicio sólo se ejecuta cuando hay algún componente asociado o enlazado (*bound*) a él.

Componentes básicos de Android

Descripción breve de los componentes básicos de Android

Broadcast Receiver: Los receptores de notificaciones o **broadcast receivers** permiten que se reciban notificaciones (y por tanto datos), desde ciertos componentes. El componente que envía la notificación lo hace a través de un *sendbroadcast*. Los componentes que la reciben se asocian o suscriben a un *Receiver*.

Content Provider: Los proveedores de contenido o **content providers** proporcionan facilidades de acceso a datos estructurados. Además de encapsular los datos, proporcionan mecanismos para definir la seguridad de los datos. Se puede considerar la interfaz que conecta los datos en un proceso con el código que se ejecuta en otro proceso. Para el intercambio de datos se usa un modelo cliente-servidor (*ContentResolver* para los clientes y *ContentProvider* para los servidores).

Primera aplicación Android: ACTIVITIES + BROADCAST RECEIVERS + SERVICES.

Ejemplo de APLICACIÓN ANDROID simple

Diseño del sistema

Especificación: Aplicación que recibe números a través de un servicio y actualiza una vista de usuario. Concretamente, se visualiza el último número generado por el servicio y el último número primo en sendas cajas de texto, ver Figura 1.

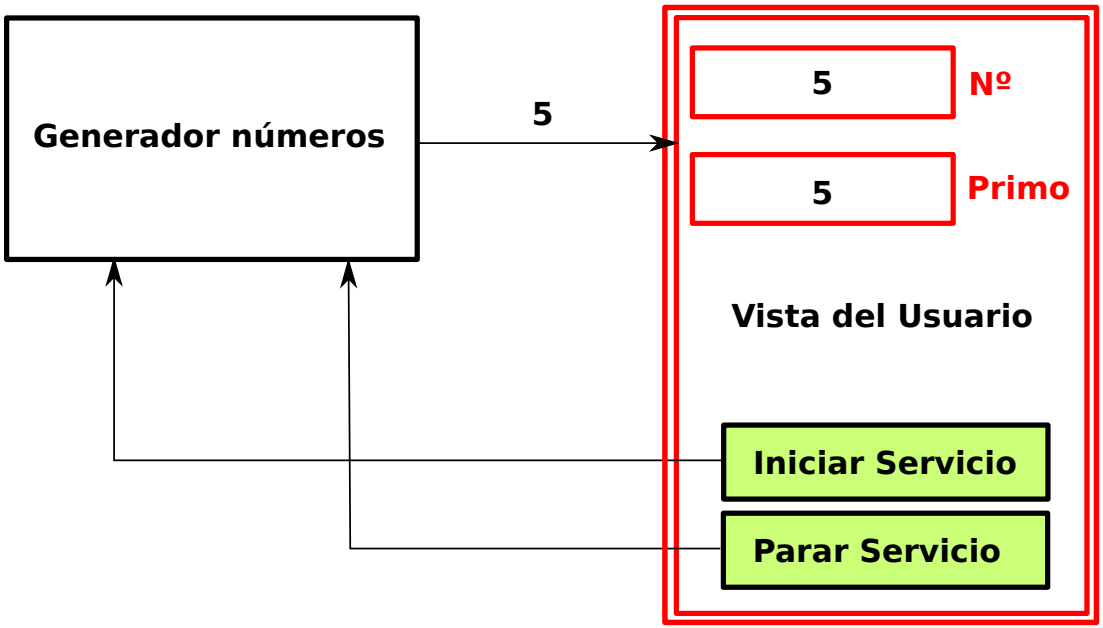

Figura 1: Diagrama de la especificación del sistema propuesto.

Ejemplo de APLICACIÓN ANDROID simple

Selección apropiada de componentes

Durante la fase de diseño, y atendiendo a la especificación del sistema, se seleccionan los componentes Android necesarios que se utilizarán en la fase de implementación.

Se trata de una decisión importante que es necesario tomar de forma cuidadosa.

La Figura 2 muestra de forma gráfica un resumen de cómo interactúan, en general, determinados componentes de Android. En nuestra aplicación se necesitan los siguientes componentes:

- **Activity**: Con una vista asociada que permita al usuario iniciar y parar el servicio y muestre los números correspondientes.
- **Service**: Un servicio que genere en background los números independientemente del comportamiento de la **activity**.
- **Broadcast Receiver**: Un receptor de mensajes que maneje el evento *Receive*. Cada vez que se recibe un número se debe procesar y actualizar convenientemente la vista asociada al **activity**. El **Broadcast Receiver** se puede definir como una atributo de la **activity**.

Ejemplo de APLICACIÓN ANDROID simple

Selección apropiada de componentes

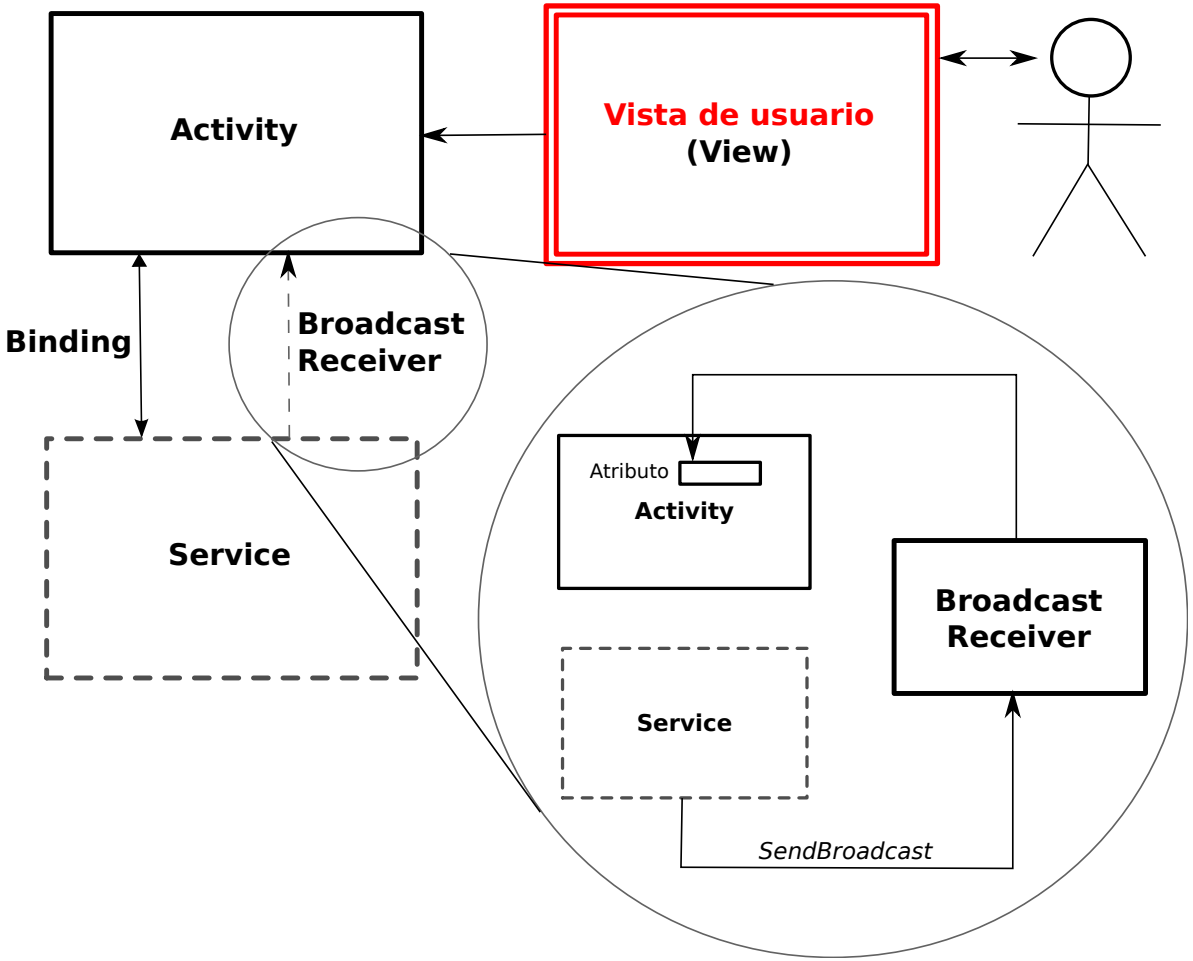

Figura 2: Interacciones entre diferentes componentes Android.

Ejemplo de APLICACIÓN ANDROID simple

Selección apropiada de componentes

La Figura 3 muestra gráficamente el conjunto de componentes Android que se necesitan para implementar la aplicación propuesta.

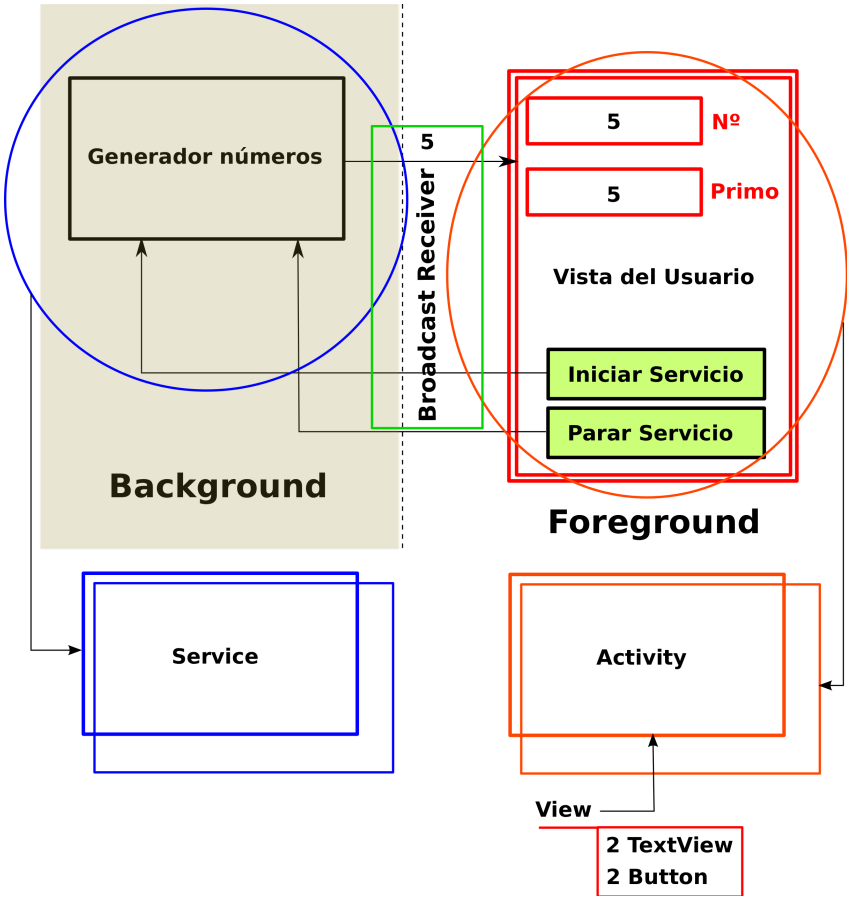

Figura 3: Conjunto de componentes Android necesarios para la aplicación.

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

Una vez que se han elegido los componentes, es necesario determinar como interactúan, se comunican y comparten información. Para ello, es necesario analizar el comportamiento de cada uno de esos componentes:

ACTIVITY: Para implementar una **activity** es necesario crear una clase que derive de la clase *Activity*. La clase *Activity* proporciona un conjunto de métodos que se ejecutan como consecuencia de la generación de diversos eventos. Atendiendo a estos eventos, una actividad pasa por diferentes estados de su ciclo de vida, ver Figura 4: Created, Started, Resumed, Running, Paused, Stopped, Restarted y Destroyed. Los métodos asociados a estos eventos son, respectivamente: *onCreate*, *onStart*, *onResume*, *onPause*, *onStop*, *onRestart* y *onDestroy*. (Obsérvese que el estado Running no está asociado a ningún evento).

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

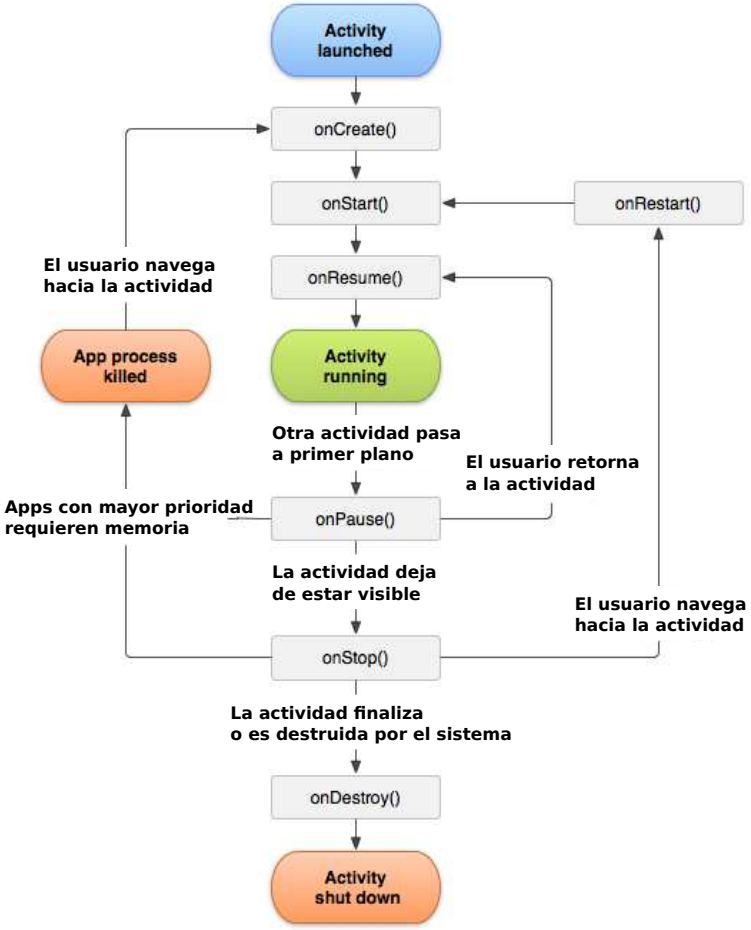

Figura 4: Ciclo de vida de un componente **activity**.

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

La actividad que muestra la interfaz de usuario actúa, además, como actividad principal. Todas los elementos de la interfaz (botones y cajas de texto) se manejan desde esta actividad principal. Por tanto, sólo es necesario escribir código (redefinir) para el método *onCreate* (donde se creará el objeto BroadcastReceiver utilizado para recibir la información proporcionada por el servicio. Dicho objeto se define como un nuevo atributo de la subclase que deriva de *Activity*). Por otra parte, la pulsación de los botones puede generar dos comportamientos distintos: iniciar o parar el servicio, que deben ser implementados como dos métodos que se activan como respuesta al evento *onClick* sobre cada botón.

Por otra parte, la actividad tiene asociada una vista de usuario o View que se define mediante un archivo *.xml*.

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

Parte del código fuente de la actividad:

```
1  public class MainActivity extends Activity{
2     private MyReceiver mReceiver;
3     ...
4
5     @Override
6     protected void onCreate(Bundle savedInstanceState) {
7        super.onCreate(savedInstanceState);
8        setContentView(R.layout.activity_main);
9        TextView t = (TextView)findViewById(R.id.textView1);
10       TextView t2 = (TextView)findViewById(R.id.textView2);
11       mreceiver = new MyReceiver(t, t2);
12    }
13    ...
14
15    public void IniciaServicio(View view){
16       ...
17    }
18
19    public void DetenerServicio(View view){
20       ...
21    }
22 }
```

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

Antes de comentar el código fuente de la actividad principal, veamos parte del código XML que define la vista de usuario:

```
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      tools:context=".MainActivity" >
6      <TextView
7          android:id="@+id/textView1"
8          ...
9          android:text="TextView" />
10     <Button
11         android:id="@+id/button1"
12         ...
13         android:onClick="IniciaServicio"
14         android:text="Iniciar Servicio" />
15     <Button
16         android:id="@+id/button2"
17         ...
18         android:onClick="DetenerServicio"
19         android:text="Parar Servicio" />
20     <TextView
21         android:id="@+id/textView2"
22         ...
23         android:text="TextView" />
24 </RelativeLayout>
```

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

Las líneas 6-9 y 20-23 del archivo XML muestran la definición de dos cajas de texto (TextView) cuyos identificadores son *"@+id/-textView1"* y *"@+id/textView2"*.

La pregunta es: ¿Cómo acceder a dichos elementos desde el código fuente de la actividad?

Las **líneas 9 y 10** del **código fuente de la actividad** permiten el acceso a dichas cajas de texto.

Observemos detenidamente el método utilizado para obtener la referencia a un *TextView*:

```
TextView t = (TextView) findViewById(R.id.textView1);
```

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

El método de la clase *Activity* *findViewById* acepta por parámetro un número entero y devuelve una referencia a un objeto de la clase *View*. Utilizando la operación de "cast" se define cuál es la clase derivada real del objeto cuya referencia es devuelta. En nuestro caso la clase es *TextView*. El identificador se pasa a través de la lista de parámetros accediendo al campo *R.id.textView1*.

La estructura *R* se rellena convenientemente cuando se ejecuta el método *onCreate* a partir del archivo XML que define la vista de usuario.

La Figura 5 muestra la organización de archivos de cualquier aplicación Android durante el proceso de implementación. (IDE de Eclipse)

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

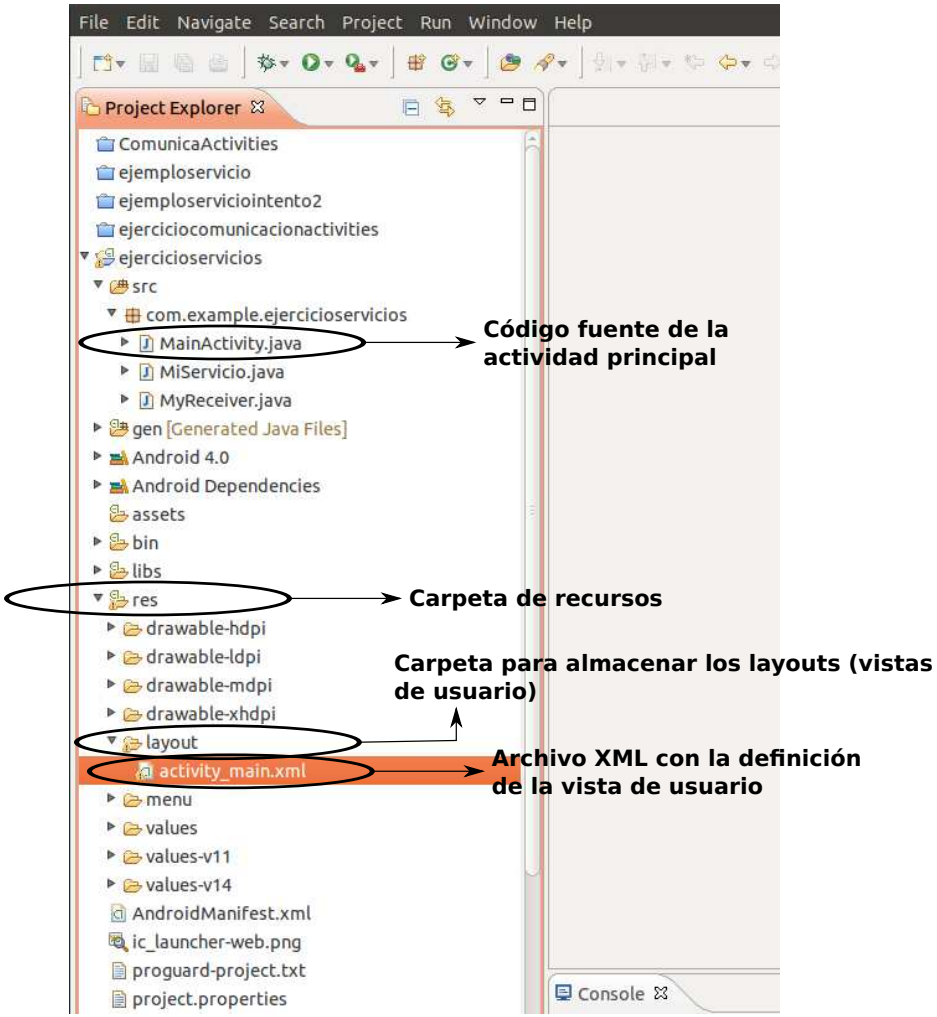

Figura 5: Organización de las carpetas que contienen el código fuente de una aplicación Android.

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

Observamos que para acceder al layout que define la vista de usuario, se usa *R.layout.activity_main*. La ruta equivalente en el árbol de directorios o carpetas que organiza el código fuente de la aplicación es *res/layout/activity_main.xml*.

Para acceder al identificador de cualquier elemento que forma parte de la vista se utiliza *R.id.IdentificadorElemento*. Por ejemplo, tal y como se indica en la línea 21 del archivo XML, si se desea acceder a la caja de texto cuyo identificador es *textView2* debemos escribir *R.id.textView2*. Si deseamos acceder al botón que permite iniciar el servicio, es necesario escribir *R.id.button1* (línea 11 del archivo XML).

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

Cuando se asignan nombres o identificadores a los elementos de una vista de usuario es convenientemente respetar el método de asignación de nombres de Android.

Cuando se desea usar una nueva cadena de caracteres para nombrar algún elemento, es habitual añadir dicha cadena al archivo *strings.xml*, que se encuentra ubicado en la subcarpeta **values** dentro de la carpeta **res**. (Una técnica similar se usa para definir estilos (styles), tamaños (size) o colores (color), entre otros. En este caso se accede al archivo *styles.xml* situado en la misma carpeta).

Por ejemplo, el nombre de la aplicación está definido como:

$< stringname = "app\_name" > nombreaplicacion < /string >$

En el archivo AndroidManifest.xml se puede observar cómo se usa el string creado para definir el nombre de la aplicación:

$< applicationandroid : allowBackup = "true" android : icon = "@drawable/ic\_launcher" android : label =$

$"@string/app\_name" android : theme = "@style/AppTheme" >$

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

Observamos que en las líneas 13 y 18, respectivamente, del archivo XML que define los elementos de la vista de usuario se especifican los nombres de los métodos de la clase actividad (derivada de *Activity*), que han de ejecutarse cuando se produce el evento *onClick* sobre cada botón:

```
android:onClick="IniciaServicio"
...
android:onClick="DetenerServicio"
```

Dichos métodos se definen como dos nuevos miembros de la clase *MainActivity* (que deriva de *Activity*). En concreto, ambos métodos reciben por parámetro un objeto de la clase *View* y no devuelven nada (*void*). Respectivamente los métodos son (líneas 15 y 19 del archivo "*MainActivity.java*" que contiene el código fuente de la actividad):

```
public void IniciaServicio (View view){...}
...
public void DetenerServicio (View view){...}
```

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

El layout o vista de usuario se ha actualizado convenientemente mediante la llamada al método *setContentView* (ver línea 8 del código fuente de "MainActivity.java"). A través de dicha llamada, es posible asociar la vista de usuario definida en el archivo "activity_main.xml" con la actividad *MainActivity*. En este caso, el parámetro pasado al método *setContentView* es de tipo entero y representa el identificador del layout o vista que debe ser visualizado. Dicho layout se almacena como un recurso (resource) de la aplicación.

Las llamadas *IniciaServicio* y *DetenerServicio* permiten que se inicie y se pare un servicio en background, cuyos resultados son capturados por un broadcast receiver. Para ello se utiliza el atributo *mReceiver* de la clase *MyReceiver* (heredada de *BroadcastReceiver*).

Por último, a través del objeto *savedInstanceState* de la clase *Bundle* (*super.onCreate(savedInstanceState)*), ver línea 6 y 7 de "MainActivity.java", es posible guardar el estado de la actividad en el caso de que haya que restaurarla si el sistema la ha destruido debido, por ejemplo, a que existían necesidades de memoria de procesos más prioritarios.

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

El servicio que se va a utilizar se ejecutará en background en su propio hilo de ejecución.

Todos los componentes de una misma aplicación Android se ejecutan en el mismo proceso Linux, salvo que se indique lo contrario, ver Figura 6. Además, todos los componentes comparten el mismo hilo de ejecución, denominado *Hilo de Ejecución Principal (HEP)*. [1]

Si un servicio creado en background realiza tareas que tengan un coste computacional elevado puede provocar un problema de *Application Not Responding (ANR)*, ya que bloquea el *HEP*. Para evitar esta situación, el servicio debe ejecutarse en un hilo secundario o *"worker thread"*.

[1] En la dirección http://developer.android.com/guide/components/processes-and-threads.html se puede encontrar información extensa sobre procesos e hilos en Android.

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

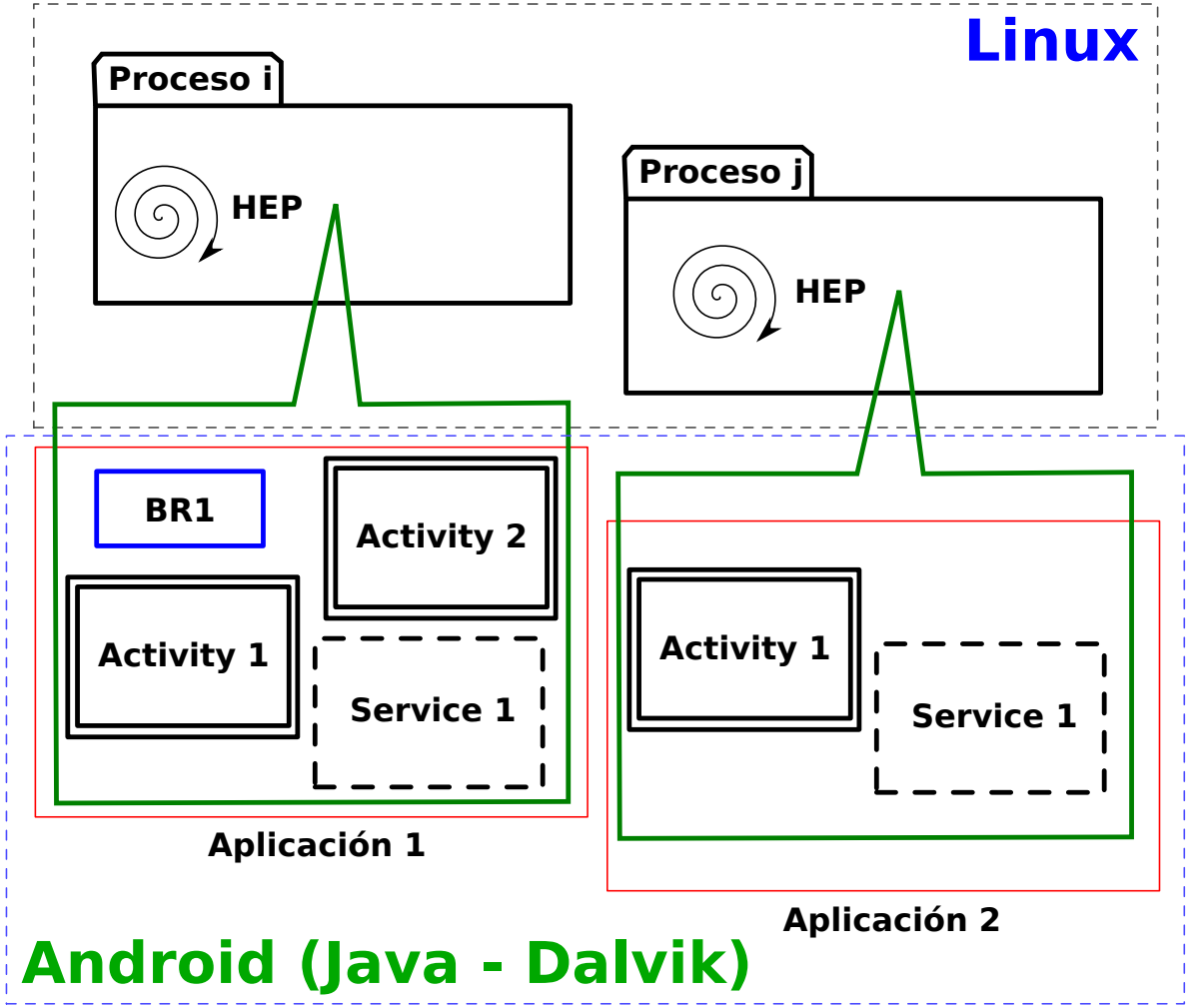

Figura 6: Relación entre componentes Android de una aplicación y su correspondiente proceso Linux.

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

Servicio en background en su propio worker thread.

Veamos el código fuente que implementa el cuerpo del método *IniciaServicio* de la actividad *MainActivity*:

```
1  public void IniciaServicio (View view){
2      contadorinicial = 0;
3      Intent i = new Intent(this, MiServicio.class);
4      i.putExtra("msg_ini_contador", contadorinicial);
5      IntentFilter filter = new IntentFilter("com.example.ejercicioservicios.recibir");
6      this.registerReceiver(mreceiver, filter);
7      Toast.makeText(this, "Voy a iniciar el servicio", Toast.LENGTH_SHORT).show();
8      startService(i);
9  }
```

Las líneas 3, 4 y 8 permiten iniciar un servicio en background. Específicamente:

- Línea 3: Creación del **Intent** que definirá la acción de iniciar el servicio cuyo código objeto se define en el archivo "MiServicio.class".
- Línea 4: Paso de datos hacia el servicio mediante el **Intent**.
- Línea 8: Inicio del servicio a través del **Intent** *i*.

El código fuente del que se ha obtenido el código objeto de "MiServicio.class" se encuentra implementado en el archivo "MiServicio.java".

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

Antes de examinar en detalle como se inicia el servicio a través de un *Intent*, veamos el código fuente de la clase que implementa dicho servicio. Dado que se desea que el servicio se ejecute en background en un *worker thread*, utilizamos la clase **IntentService** como clase base a partir de la cual heredará nuestra clase *MiServicio*.[2]

```
1  public class MiServicio extends IntentService {
2     private int contador = 0;
3     private boolean mifinal = false;
4     public static final String ACCION = "com.example.ejercicioservicios.recibir";
5     public MiServicio() {super("MiServicio");}
6     @Override   public IBinder onBind(Intent intent) {...}
7     @Override
8     public int onStartCommand(Intent intent, int flags, int startId){
9        contador = Integer.parseInt(intent.getExtras().
10       get("msg_ini_contador").toString());
11       mifinal = false;
12       return (super.onStartCommand(intent, flags, startId));
13    }
14    @Override
15    protected void onHandleIntent(Intent intent) {
16       while (!mifinal){
17          Intent i = new Intent(ACCION);
18          i.putExtra("valorcontador", contador);
19          sendBroadcast(i);
20          SystemClock.sleep(1000);
21          contador++;
22       }
23    }
24    @Override   public void onDestroy(){...}
25 }
```

[2] Si no se desea que el servicio se ejecute en un thread distinto, es posible utilizar la clase *Service* como clase base.

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

Cuando se implementa un servicio como una clase derivada de *IntentService* es necesario redefenir los siguientes métodos:

- *onStartCommand*: Se ejecuta cuando se inicia el servicio. Es adecuado utilizarlo para definir procesos de inicialización. Por ejemplo, en la aplicación se obtiene el valor inicial de contador que el servicio incrementará a través del *Intent* utilizado por la actividad que lo llamó.
- *onHandleIntent*: Este método ejecuta el conjunto de instrucciones que permiten proporcionar el servicio a través de un hilo diferente al hilo principal (worker thread).

Además, es obligatorio definir el método *public MiServicio()*, simplemente llamando a *super("MiServicio")*.

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

El método *onStartCommand* puede retornar valores diferentes. En este caso el valor retornado es aquel que devuelve la llamada a *super.onStartCommand(Intent, flags, startId)*. En general, el método *onStartCommand* puede devolver las siguientes constantes:

- **START_STICKY**: Este valor debería ser retornado por servicios que explícitamente son iniciados o parados para llevar a cabo una tarea durante un período de tiempo arbitrario. Por ejemplo, un servicio que reproduce música en background. Si el sistema mata al proceso en el que se ejecuta el servicio después de ser iniciado, se guarda el estado (concretamente iniciado), pero no se retiene el *Intent* despachado. Cuando el sistema recrea el proceso que contiene al servicio, entonces se vuelve a producir el evento *onStart* y por tanto se ejecuta *onStartCommand*. Si no quedaban comandos de inicio pendientes de ser despachados al servicio, entonces el servicio se crea pasando un objeto *Intent* nulo.
- **START_REDELIVER_INTENT**: En este caso, si el proceso que alberga al servicio es eliminado (killed), cuando el servicio ya se ha iniciado, dicho servicio es planificado para un reinicio y el último *Intent* es despachado de nuevo mediante una llamada a *onStartCommand*. De hecho, el *Intent* permanecerá planificado para ser despachado de nuevo hasta que el servicio llame a *stopSelf*. De este modo, hasta que el servicio no ha ejecutado la acción definida en el *Intent*, dicho *Intent* no es eliminado. Los servicios que se inician devolviendo este valor no pueden ser recreados mediante un *Intent* nulo, ya que el sistema únicamente los recrea si quedan peticiones pendientes o la última petición no se ejecutó completamente.
- **START_STICKY_COMPATIBILITY**: Se trata de una versión de compatibilidad con el modo *START_STICKY* que no garantiza que se llame a *onStartCommand* cuando el servicio es recreado una vez que su proceso ha sido eliminado.
- **START_NOT_STICKY**: En este caso, si el sistema mata al proceso que permite la ejecución del servicio cuando ya se ha iniciado dicho servicio, y no existen nuevos *Intent* para ser despachados, el servicio se considera no iniciado y no se recrea hasta que explícitamente se haga una llamada a *Context.startService(Intent)*. Donde *Context* representa el componente desde donde se inicia el servicio, por ejemplo, una actividad.

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

El contexto desde el cual se inicia el servicio es la actividad *MainActivity*. Tal y como se indica en las líneas 3, 4 y 8 del método *IniciaServicio* del archivo *MainActivity.java*, el servicio cuyo código objeto se encuentra almacenado en el archivo *MiServicio.class*, es iniciado a través de un objeto *Intent*.

Los objetos de la clase **Intent** se utilizan para especificar acciones que se desean llevar a cabo. Por ejemplo, iniciar una nueva actividad desde otra actividad es una acción que se debe realizar mediante un **Intent**. En este caso hablamos de *Intent* explícito, ya que especificamos qué componente debe ejecutarse.

Supongamos que deseamos iniciar una actividad *B* cuya funcionalidad está definida en un archivo denominado *B.java*, y cuyo código objeto está almacenado en el fichero *B.class*. La actividad desde la cual se realiza dicha inicialización se denomina *A*. En uno de los métodos de *A* debemos escribir el código de la transparencia siguiente.

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

Es posible, además, pasar información desde A a B usando un sistema de paso de mensajes que admite el *Intent*.

```
Metodo de actividad A
...
Intent i = new Intent(this, B.class);
i.putExtra("identificadormensaje", mensaje);
startActivity(i);
...
```

Los mensajes pasados a través de un *Intent*, se identifican mediante un *String* que actúa como identificador del mensaje. El mensaje en sí puede ser un *String*, un entero, etc.

En lugar de hacer un *Intent* explícito, es posible solicitar una acción a través de un *Intent* implícito. En este caso, no se especifica el componente que ejecutará la acción sino la acción que debe ser ejecutada. El sistema seleccionará el componente más adecuado para llevar a cabo dicha acción. Para ello, examina los *Intent Filter* que determinan qué acciones pueden ser respondidas por un determinado componente. En el fichero "AndroidManifest.xml" se definen los *Intent Filter* para cada componente implicado en una determinada aplicación.

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

Del mismo modo que para iniciar una nueva actividad se usa un *Intent* explícito, para poner en marcha un servicio concreto, también usamos un *Intent* explícito. A continuación se muestra nuevamente el código del método *IniciaServicio* y el código de *DetenerServicio*.

```
1  public void IniciaServicio (View view){
2    contadorinicial = 0;
3    Intent i = new Intent(this, MiServicio.class);
4    i.putExtra("msg_ini_contador", contadorinicial);
5    IntentFilter filter = new IntentFilter("com.example.ejercicioservicios.recibir");
6    this.registerReceiver(mreceiver, filter);
7    Toast.makeText(this, "Voy a iniciar el servicio", Toast.LENGTH_SHORT).show();
8    startService(i);
9  }
10 public void PararServicio (View view){
11   Intent i = new Intent(this, MiServicio.class);
12   this.unregisterReceiver(mreceiver);
13   Toast.makeText(this, "Voy a parar el servicio", Toast.LENGTH_SHORT).show();
14   stopService(i);
15 }
```

Como indican las líneas 3, 4 y 8, la actividad inicia el servicio y además le pasa el valor inicial del contador a través del sistema de mensajes del *Intent*. Las líneas 11 y 14 muestran como parar el servicio. Para ello se solicita dicha acción a través de un *Intent* explícito.

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

Cuando se crea el servicio, se produce el evento *onCreate*. Esto provoca que se llame al método *onCreate()*. Por tanto, si deseamos que cuando se cree el servicio se ejecuten determinadas acciones, se puede redefinir dicho método. Obviamente, la creación del servicio se produce antes de que se inicie mediante *onStartCommand* o mediante *onBind* (para *bound services*).

Cuando el servicio no se usa más, puede ser destruido por el sistema mediante una llamada al método *onDestroy()*. Este método puede redefinirse para liberar todos los recursos y estructuras que haya usado el servicio. De hecho, ésta es la última llamada que recibe un servicio antes de ser destruido.

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

La Figura 7 representa el ciclo de vida de un servicio cuando es iniciado (started) a partir de otro componente, por ejemplo, una actividad, mediante la llamada al método *startService*.

Figura 7: Ciclo de vida de un servicio started.

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

La Figura 8 muestra un resumen de como interactúan la actividad, el **Intent** y el servicio cuando se produce el inicio y parada del mismo.

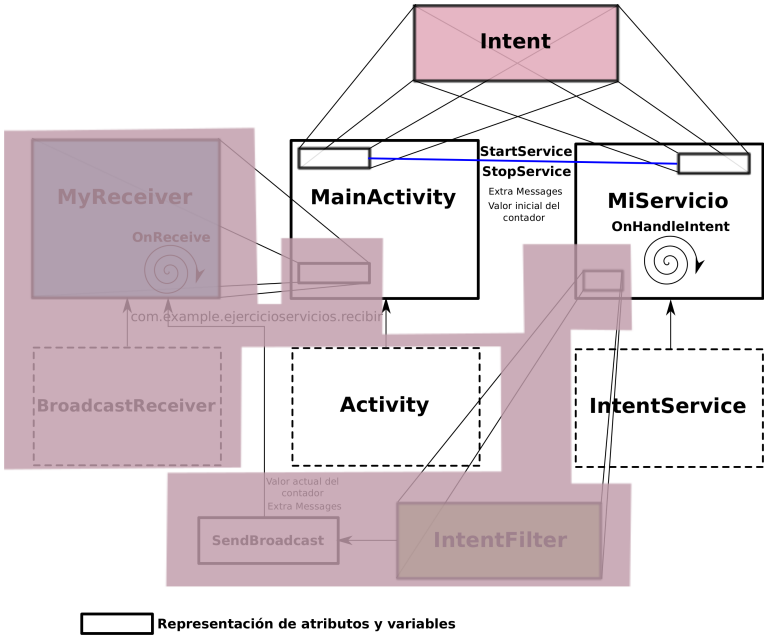

Figura 8: Interacción de los principales componentes de la aplicación de ejemplo. Detalle de la interacción del servicio, el **Intent** y la actividad.

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

Los resultados proporcionados por el servicio deben llegar de nuevo a la actividad. Para ello se utiliza un componente de tipo **broadcast receiver**.

Cada vez que el servicio genera un nuevo dígito, envía una notificiación o mensaje de broadcast utilizando el método *sendBroadcast*. Dicho método acepta por parámetro un objeto de tipo *Intent* o *IntentFilter*. En este caso, se usa un IntentFilter, de modo que especificamos qué elementos pueden recibir la notificación.

Se desea que el mensaje sea recibido por un componente de tipo **broadcast receiver**. En nuestro caso, usamos *MyReceiver* como clase derivada de *BroadcastReceiver* que permite implementar el receptor. Dicho receptor será capaz de recibir los mensajes emitidos por otro componente a través de un *IntentFilter* cuya acción está determinada por la constante *com.example.ejercicioservicios.recibir*. El nombre de esta constante ha sido seleccionado de forma personalizada.

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

En la actividad *MainActivity* existe un atributo de la clase *MyReceiver*. Cuando se desea que el receptor esté preparado, es necesario registrar dicho receptor mediante una llamada a *registerReceiver*. Cuando se hace el registro del receptor también se especifica (de forma dinámica), la acción que provocará que dicho receptor dispare su evento *onReceive*. En este caso, la acción se ha definido de forma personalizada mediante el string *com.example.ejercicioservicios.recibir*:

```
1  public void IniciaServicio (View view){
2    ...
3    IntentFilter filter = new IntentFilter("com.example.ejercicioservicios.recibir");
4    this.registerReceiver(mreceiver, filter);
5    ...
6  }
7  public void PararServicio (View view){
8    Intent i = new Intent(this, MiServicio.class);
9    this.unregisterReceiver(mreceiver);
10   ...
11 }
```

Obsérvese que para desregistrar el receptor sólo se necesita llamar a *unregisterReceiver* pasando como argumento *mreceiver*, que no es más que el atributo de tipo *MyReceiver* declarado en la *MainActivity*.

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

El código del receptor *MyReceiver* está implementado en *MyReceiver.java* y se muestra a continuación:

```
1  public class MyReceiver extends BroadcastReceiver {
2     private TextView t, t2;
3     public MyReceiver(TextView t, TextView t2) {
4        this.t = t;
5        this.t2 = t2;
6        t.setText(" ");
7        t2.setText(" ");
8     }
9
10    public boolean esprimo(int n){
11       int i;
12       for (i = 2; i < n; i++){
13          if (n%i == 0) return false;
14       }
15       return true;
16    }
17
18    @Override
19    public void onReceive(Context context, Intent intent) {
20       String s = intent.getExtras().get("valorcontador").toString();
21       t.setText(s);
22       if (esprimo(Integer.parseInt(s)))
23          t2.setText(s);
24     }
25 }
```

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

Cada vez que el servicio genera un nuevo número, envía un mensaje de broadcast a través de un *IntentFilter* cuya acción es *com.example.ejercicioservicios.recibir*. Dado que el receptor se registró para que recibiese mensajes enviados a través de un *IntentFilter* donde se ha especificado dicha acción, el receptor *MyReceiver* disparará el método *onReceive* cada vez que se reciba un mensaje. Obtendrá a partir de los datos extras del *Intent* el valor del contador y actualizará dos *TextView*.

Uno de los *TextView* se actualiza siempre que se recibe un mensaje (línea 21), y el otro *TextView* cuando el número recibido es primo (líneas 22 y 23). El método *esprimo* se implementa directamente en la clase, aunque podría tratarse de un método de otra clase definido en otro paquete.

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

Cuando se llama al constructor de la clase *MyReceiver* se pasan dos parámetros de la clase *TextView*. Los elementos de la vista que muestran la secuencia de números del contador, así como los primos, se implementan mediante dos *TextView*. Desde la clase *MyReceiver* se accede, concretamente, a estos elementos ya que el contenido de los atributos *t* y *t2* es la misma referencia a los objetos *TextView* mostrados en la vista de usario.

Nótese que en la actividad principal *MainActivity* se llama al constructor de *MyReceiver* en el método *onCreate*:

```
...
MyReceiver(t, t2)
...
```

Donde *t* y *t2* son dos variables locales de tipo *TextView* que han sido inicilizadas usando las sentencias:

```
TextView t = (TextView)findViewById(R.id.textView1);
TextView t2 = (TextView)findViewById(R.id.textView2);
```

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

La Figura 9 muestra un resumen de como interactúan todos los componentes de la aplicación.

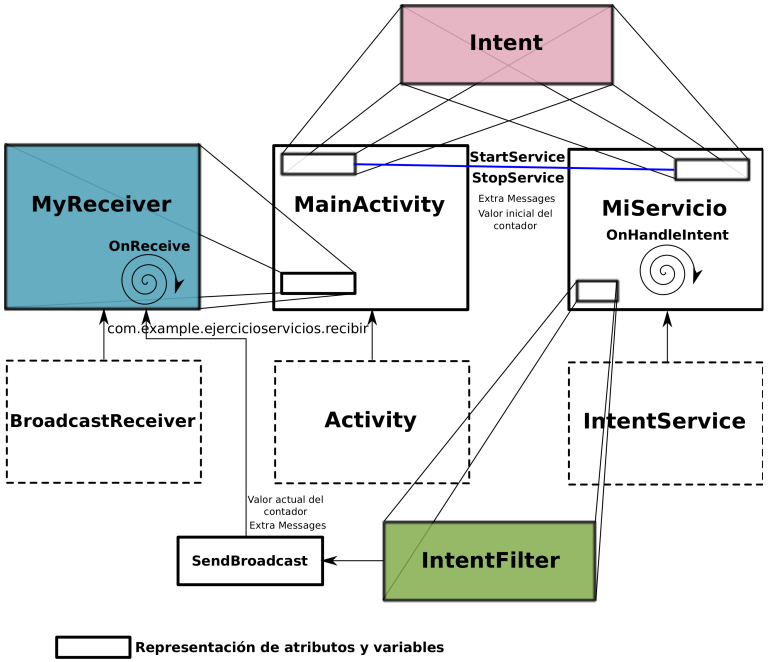

Figura 9: Interacción de los principales componentes de la aplicación de ejemplo.

Ejemplo de APLICACIÓN ANDROID simple

Análisis de los componentes Android utilizados: Activity, Service y BroadcastReceiver

Cuando se especifica una petición para que el servicio se pare mediante la pulsación del botón de la actividad principal correspondiente, se ejecuta el método *onDestroy* del servicio. En este punto, el atributo *mifinal* se pone a *false*. Justo cuando dicha variable cambia de estado, el bucle del método *onHandleIntent* termina y el hilo trabajador se detiene, ya que los servicios creados heredando de *IntentService* realizan automáticamente la operación *selfStop* cuando finaliza la ejecución del método *onHandleIntent*.

Ejemplo de APLICACIÓN ANDROID simple

Fichero AndroidManifest.xml

El fichero *AndroidManifest.xml* permite especificar los componentes que forman la aplicación, así como otros elementos importantes relacionados con la configuración de la misma. En el ejemplo, el fichero *AndroidManifest.xml* utilizado se compone de las siguientes sentencias:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="com.example.ejercicioservicios"
4     android:versionCode="1"
5     android:versionName="1.0" >
6     <uses-sdk
7        android:minSdkVersion="8"
8        android:targetSdkVersion="14" />
9     <application
10       android:allowBackup="true"
11       android:icon="@drawable/ic_launcher"
12       android:label="@string/app_name"
13       android:theme="@style/AppTheme" >
14       <activity
15          android:name="com.example.ejercicioservicios.MainActivity"
16          android:label="@string/app_name" >
17          <intent-filter>
18             <action android:name="android.intent.action.MAIN" />
19             <category android:name="android.intent.category.LAUNCHER" />
20          </intent-filter>
21       </activity>
22       <service
23          android:name="com.example.ejercicioservicios.MiServicio"
24          android:enabled="true"
25          android:exported="true" ></service>
26    </application>
27 </manifest>
```

Ejemplo de APLICACIÓN ANDROID simple

Fichero AndroidManifest.xml

Las líneas del fichero *AndroidManifest.xml* que definen el conjunto de intents que puede recibir la aplicación:

```
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
```

Indican que la actividad correspondiente es la actividad principal. Este filtro es necesario si deseamos que la actividad se inicie cuando se inicia la aplicación. Además indicamos en *"category"* (categoría) que dicha actividad puede ser lanzada por el sistema.