Laboratorio 5 – Introducción a la programación Android Wear

Sergio Daniel Hernandez
Laura Avila

26/04/2016

1. **Estructura de proyectos**
    a. Elementos del API de Android Wear y su función
        i. wearable.DataApi: Expone un API con componentes para leer o escribir data items o assets.
        ii. wearable.DataEvent: Interface de datos para los eventos de datos
        iii. wearable.DataEventBuffer: Estructura de datos que hace referencia a un set de eventos
        iv. wearable.MessageApi: Expone una API de componentes para enviar mensajes a otros nodos
        v. wearable.MessageEvent: Información acerca de los mensajes recibidos por un listener
        vi. wearable.Node: Información acerca de un nodo específico en la red del Android Wear.
        vii. wearable.NodeApi: Expone un API para obtener información de nodos locales o conectados
        viii. wearable.WearableListenerService: Recibe eventos de otros nodos, como cambios en los datos, mensajes o eventos de conectividad.
        ix. wearable.WearableStatusCodes: Codigos de error para fallas en el API del wearable
        x. wearable.PutDataRequest: Se usa para crear nuevos data items en la red de Android Wear
        xi. wearable.PutDataMapRequest: Es una versión de PutDataRequest pero para DataMap
        xii. wearable.Wearable: Un API para la plataforma de Android Wear
        xiii. wearable.DataMap: Un mapa de datos soportada por PutDataMapRequest y DataMapItems
        xiv. wearable.DataMapItem: Crea un nuevo data item como un objeto que contiene información estructurada y serializable.

2. **Despliegue en emulador**
    a. Explicación del bloque de código en el módulo mobile que logra que dicha notificación sea enviada:

```
private void buildWearableOnlyNotification(String title, String content, String path) {
    if (mGoogleApiClient.isConnected()) {
        PutDataMapRequest putDataMapRequest = PutDataMapRequest.create(path);
        putDataMapRequest.getDataMap().putString(Constants.KEY_CONTENT, content);
        putDataMapRequest.getDataMap().putString(Constants.KEY_TITLE, title);
        PutDataRequest request = putDataMapRequest.asPutDataRequest();
        Wearable.DataApi.putDataItem(mGoogleApiClient, request)
                .setResultCallback(new ResultCallback<DataApi.DataItemResult>() {
                    @Override
                    public void onResult(DataApi.DataItemResult dataItemResult) {
                        if (!dataItemResult.getStatus().isSuccess()) {
                            Log.e("Test", "buildWatchOnlyNotification(): Failed to set the data, "
                                    + "status: " + dataItemResult.getStatus().getStatusCode());
                        }else{
                            Log.e("Test", "buildWatchOnlyNotification(): Data sent "
                                    + "status: " + dataItemResult.getStatus().getStatusCode());
                        }
                    }
                });
    } else {
        Log.e("Test", "buildWearableOnlyNotification(): no Google API Client connection");
    }
}
```

El código crea un PutDataMapRequest e ingresa los datos que obtuvo de getDataMap y crea una solicitud de tipo PutDataRequest, esta última se ingresa en el DataApi como putDataItem. Si dicho dataItem es exitoso significa que se envió la notificación.

3. Manejo de eventos

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == SPEECH_REQUEST_CODE && resultCode == RESULT_OK) {
        List<String> results = data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
        String spokenText = results.get(0);
        Log.d("DEBUG", "Google speech result: " + spokenText);
        if(spokenText.equalsIgnoreCase(Constants.ACTION_BUY_NOW))
        {
            try {
                JSONObject json = new JSONObject();
                json.put("action", Constants.ACTION_BUY_NOW);
                Log.e("ATENCION", "Enviando-> ");
                new SendMessage().execute(json.toString());

            } catch (JSONException e) {
                e.printStackTrace();
                Toast.makeText(getApplicationContext(), "Something went wrong",
                        Toast.LENGTH_LONG).show();
            }

        }else{
            Toast.makeText(getApplicationContext(), "Dijiste: " + spokenText +
                    ". Prueba \"comprar ahora\".",
                    Toast.LENGTH_LONG).show();
        }
    }
    super.onActivityResult(requestCode, resultCode, data);
}
```

a. Explicación del JSON

El objeto JSON se crea como tipo put y se le asignan los parámetros, de tipo "*action*" y la constante definida la clase Constants que equivale a la acción de comprar. Se trata de una constante de texto (String) cuyo texto debe ser igual al del comando dicho por el usuario. En este caso es: `comprar ahora` Luego se crea un nuevo sendMessage del mismo Main Activity que usa el MessageAPI para enviar el mensaje a cada uno de los nodos.

b. ¿Cuál es el método en el módulo mobile que recibe dicho JSON? ¿Cómo hace parsing del mismo?

El método que recibe estos mensajes es onMessageReceived y se encuentra en WearListenerService. El parsing se hace a un String de la siguiente manera:

```
String message = new String(messageEvent.getData());
```

Una vez recibido el mensaje, le extrae el mensaje ya como tal dado su tipo (en este caso *action*): `json.getString("action")`

Y finalmente, compara este último con la constante definida en la clase Constants (en el paquete *helper*s). Podemos reconocer en esta clase la misma constante con el mismo valor que en *wear*.

c. ¿Qué decisiones toma el módulo mobile cuando llega un action distinto a Constants.ACTION_BUY_NOW. Mencione dichos otros comandos.

Otro action distinto es Constants.ACTION_PROCESSED_PRODUCT una vez recibido se procede a obtener del JSON el id del producto y a obtener la instancia de recetas efectivas para buscar y borrar la cantidad del ingrediente con dicho id.

Una vez hecho esto se obtiene del JSON el resultado que puede ser de diferentes tipos, el primero: Constants.STATUS_BUY, en este caso se agrega en el mundo a la lista de comprados la cantidad del ingrediente. El otro resultado posible es Constants.STATUS_MISS en cuyo caso se ejecuta el método addmissing del mundo y se le envía por parámetro la cantidad del ingrediente según su id.

d. Explique cuál es el propósito del BroadcastReceiver declarado en la clase MainActivity del módulo wear en el intercambio de mensajes.

Recibe intents de los eventos de *product* (producto), al recibirlo crea un nuevo intent y le ingresa los argumentos usando un action de las *constants* (constantes) definidas Constants.ARGUMENTS para luego usar el startActivity para desplegar una Actividad

Adición de nuevos requerimientos

Para adicionar el requerimiento adicional se deben realizar varios pasos.

1. Deshabilitar los gestos que vienen por defecto (windowSwipeToDismiss).
2. Usar correctamente el GestureDetector que ofrece android.

Se usó como guía la siguiente entrada en *StackOverflow*: http://stackoverflow.com/questions/27524418/how-to-implement-gesture-recognition-in-android-wear