Tarea 6

Ensayo video

15/03/2018

Mónica Elizabeth Alba González

En la programación imperativa, si existen métodos asíncronos, se vuelve más complejo. Si se trata de acceder a una base de datos, por ejemplo, no se puede garantizar que todo el proceso que le sigue se ejecute correctamente, ya que el código no espera a que se tenga una respuesta del servidor o de la base de datos.

RxJava quita la responsabilidad del código y conecta a los elementos los unos a los otros, en lugar de manejarlos manualmente.

Las interacciones con el usuario se vuelven observables, para así reaccionar únicamente cuando se necesita. Un "Completable" notifica si el método fue realizado con éxito o fracaso una vez que haya terminado (otra manera de hacer reactivos los métodos)

"Disposables" -> los objetos se desconectan de los elementos que se utilizan cuando se haya acabado el proceso para el cua fueron requerido.

"Push-based updates" -> cuando se hayan hecho cambios en la base de datos reacciona de una manera.

El video expone un ejemplo en el cual el usuario ingresa un nombre en un celular y después pulsa un botón. Propone implementar un intermediario, que es un estado, que intercede entre el usuario y los modelos que se implementan para hacer reactiva las acciones que realiza este. Esto evita que si un método está en proceso, todas las acciones que realice el usuario durante este, sean omitidas; es decir, que dichas acciones no interrumpan el proceso en curso.

Los estados son modificados cada vez que ocurre un evento y se obtiene un callback.

En resumen: las interacciones del usuario las recibe un UiEvent que es de tipo Observable. Todas las interacciones Ui se juntan en un solo stream con el que se puede reaccionar. Del lado del sistema operativo o de algún servicio o base de datos se tiene otro observable donde se manejan las acciones síncronas y asíncronas que se solicitan. Los resultados de las acciones se controlan en otro Observable que es reusable y se maneja solamente en un stream. Estos resultados son regresados a un UiModel que hace cambios mínimos de acuerdo si los resultados son favorables o si hubo errores en los procesos.

Además se agregan los datos de estado al modelo. Esto sirve para que si el programa se interrumpe, el modelo se queda intacto y se vuelve a conectar una vez que el programa se reactive. Así no se pierde el progreso que ya se había alcanzado. Cada parte del sistema es reusable, o incluso el sistema mismo. Lo que se puede catalogar como un patrón.