LA RUEDA:

Optimización de viajes por trabajo

Anexo 4: Diseño del sistema.

Trabajo de Fin de Máster

Máster Universitario en Ingeniería Informática

VNiVERSiDAD D SALAMANCA

CAMPUS DE EXCELENCIA INTERNACIONAL

SEPTIEMBRE 2021

AUTOR:

Marcos Unzueta Puente

TUTOR:

Pablo Chamoso Santos

Índice

Índice de ilustraciones

1. Introducción

En este anexo se tratará de seguir refinando el proyecto que se está trabajando, siendo el último que trata enteramente de ingeniería del software.

Se utilizará la información obtenida en el análisis del sistema llevado a cabo en el Anexo 3 para elaborar el dominio de la solución.

Se profundizará sobre el patrón arquitectónico escogido, se realizarán diagramas de las clases de diseño teniéndolo en cuenta, se realizará un diagrama que muestre la vista de arquitectura, se refinarán los diagramas de secuencia realizados en el anexo anterior llevándolos a un nivel de abstracción más bajo y se realizará un diagrama de la vista de despliegue.

2. Modelo de diseño

Usando el modelo de análisis desarrollado en el Anexo 3, se procede a describir en un nivel de abstracción más baja, la estructura del sistema y como se implementará.

Es decir, se sigue refinando la información obtenida de los anexos anteriores. (IBM, s.f.)

En primer lugar, se procede a hablar del patrón arquitectónico de construcción elegido.

2.1. Patrón elegido

Tras un análisis exhaustivo, se decidió utilizar el patrón Modelo-Vista-Controlador por una serie de motivos que se explicarán posteriormente.

Es importante tener en cuenta que, debido a la popularidad de este patrón, existen variantes. En este documento se va a considerar la más común.

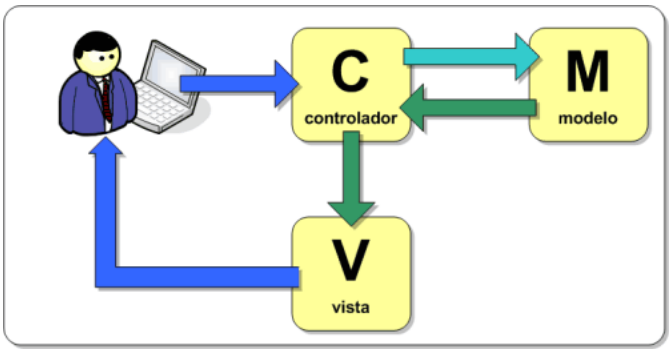

Ilustración 1: Patrón Modelo-Vista-Controlador

El principio básico de este paradigma, que ayuda a organizar y estructurar los elementos del sistema software a desarrollar, consiste en separar los componentes de una aplicación en tres capas principales (campusmvp, s.f.):

2.1.1. Vista

La vista se encarga de generar la interfaz con la que se comunicará el usuario.

En el caso concreto de este proyecto, la vista está formada por los archivos con la extensión xml que se encuentran en el directorio res.layout del proyecto y forman la interfaz de la aplicación.

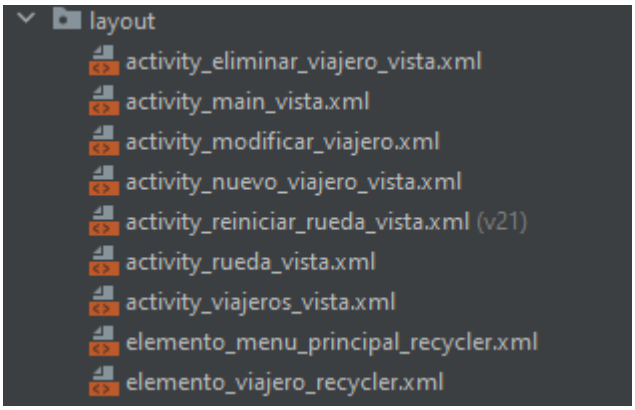

Ilustración 2: Directorio res.layout

Al programar los ficheros xml, solo se desarrolla la parte visual, en ningún momento se profundiza en las funcionalidades de la aplicación.

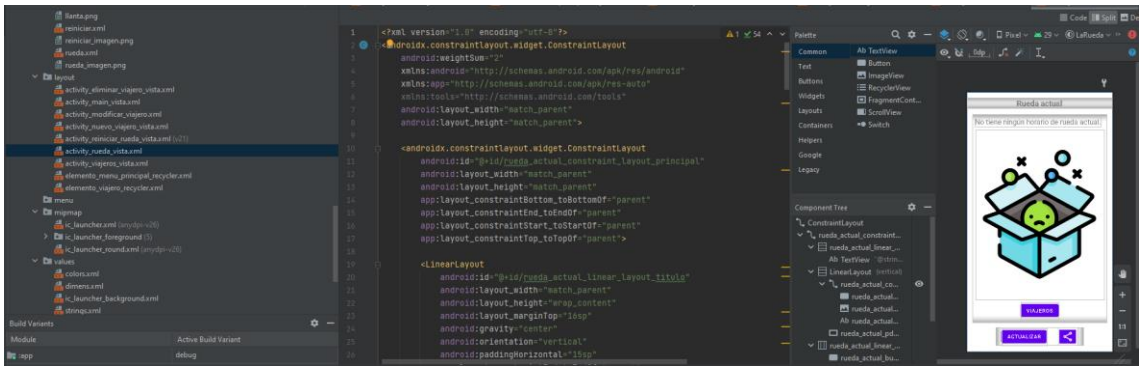

Ilustración 3: Programación del XML

La vista ha de contener los elementos que permitan al usuario interactuar y enviar información, de tal manera que este pueda solicitar la realización de acciones del sistema.

2.1.2. Controlador

La función del controlador es actuar como intermediario entre el propio usuario y el sistema. Ha de ser capaz de reconocer las interacciones sobre la vista de la aplicación, como la pulsación de un botón o el deslizamiento en un slidebar.

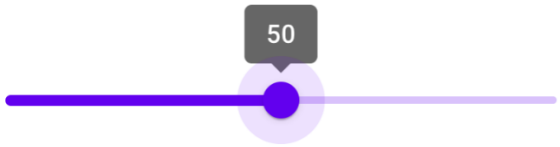

Ilustración 4: Slidebar

(material.io, s.f.)

Cuando el usuario dispare acciones del sistema interactuando con elementos de la vista, el controlador será el encargado de coordinar la respuesta, que puede ser desde cargar una nueva vista hasta solicitar al modelo que realice acciones.

En el caso de la aplicación desarrollada, es el encargado de reconocer los elementos XML de la vista, así como las acciones que el usuario lleva a cabo sobre ellos, respondiendo tanto en el tratamiento de datos (modelo) como en la información que se muestra (vista).

Se ha elegido utilizar Java como lenguaje para los controladores de la aplicación. Estos se encontrarán en subdirectorios de la carpeta java.com.marcosunzueta.larueda y están indicados por la palabra controlador precediendo al nombre del fichero.

Otra misión importante que tienen es la de ejercer de traductor entre modelo y vista, interpretando los datos que se obtienen la vista y traduciéndolos para el modelo. También se realiza el proceso inverso.

En la imagen mostrada a continuación se podrá ver un fragmento del código del controlador de la actividad encargada de añadir un nuevo viajero. Se aprecia como el controlador identifica los elementos que forman el formulario que se muestra en la vista para poder comunicar la información introducida por el usuario con el modelo.

```java
public class NuevoViajeroControlador extends AppCompatActivity {

    //Spinner correspondientes a las horas de entrada y salida que marcarán el horario del viajero.
    private MaterialSpinner spinnerLunesEntradaHoras, spinnerLunesEntradaMinutos, spinnerMartesEntradaHoras, spinner
    private MaterialSpinner spinnerLunesSalidaHoras, spinnerLunesSalidaMinutos, spinnerMartesSalidaHoras, spinnerMar

    //Adaptadores que instanciarán los valores a los spinner.
    private ArrayAdapter<String> adaptadorSpinnerHoras, adaptadorSpinnerMinutos, adaptadorVacio;
    private final NuevoViajeroModelo modelo = new NuevoViajeroModelo( controlador: this);

    private EditText nombre;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_nuevo_viajero_vista);

        spinnerLunesEntradaHoras = findViewById(R.id.nuevo_viajero_spinner_lunes_entrada_hora);
        spinnerLunesEntradaMinutos = findViewById(R.id.nuevo_viajero_spinner_lunes_entrada_minuto);
        spinnerMartesEntradaHoras = findViewById(R.id.nuevo_viajero_spinner_martes_entrada_hora);
        spinnerMartesEntradaMinutos = findViewById(R.id.nuevo_viajero_spinner_martes_entrada_minuto);
        spinnerMiercolesEntradaHoras = findViewById(R.id.nuevo_viajero_spinner_miercoles_entrada_hora);
        spinnerMiercolesEntradaMinutos = findViewById(R.id.nuevo_viajero_spinner_miercoles_entrada_minuto);
        spinnerJuevesEntradaHoras = findViewById(R.id.nuevo_viajero_spinner_jueves_entrada_hora);
        spinnerJuevesEntradaMinutos = findViewById(R.id.nuevo_viajero_spinner_jueves_entrada_minuto);
        spinnerViernesEntradaHoras = findViewById(R.id.nuevo_viajero_spinner_viernes_entrada_hora);
        spinnerViernesEntradaMinutos = findViewById(R.id.nuevo_viajero_spinner_viernes_entrada_minuto);
        spinnerSabadoEntradaHoras = findViewById(R.id.nuevo_viajero_spinner_sabado_entrada_hora);
        spinnerSabadoEntradaMinutos = findViewById(R.id.nuevo_viajero_spinner_sabado_entrada_minuto);
```

Ilustración 5: Código controlador

2.1.3. Modelo

En el modelo se encuentran los datos que maneja el sistema, así como la lógica del negocio.

En el caso de esta aplicación Android llevada a cabo en el entorno de desarrollo integrado Android Studio, se utilizan ficheros programados con el lenguaje java, que almacenarán la información en atributos y operarán haciendo uso de métodos creados.

Una función importante del modelo de esta aplicación es la de gestionar la interacción de la aplicación con la base de datos que almacenará la información. (androidauthority, s.f.)

Ilustración 6: Android SQLite

2.2. Ventajas y desventajas

Como es lógico, el patrón utilizado tiene una serie de ventajas y desventajas que van a afectar directamente al proyecto. En este subapartado se procede a enumerar las mismas.

También se explicará cómo afecta cada ventaja al proyecto concreto, siendo este estudio el que decantó la elección del patrón.

2.2.1. Ventajas:

- La separación definida de funciones en los distintos elementos de la aplicación provoca que partes independientes del software se puedan modificar y actualizar de manera independiente. Esto provoca que la elaboración de este proyecto pueda ser más incremental.

- El desarrollo será más limpio y sencillo, algo muy positivo en un trabajo universitario, ya que para su posterior explicación delante del tribunal es mucho más atractivo un código limpio y bien estructurado.

- La depuración de errores es relativamente fácil ya que el código es más modular, lo cual permite localizar los errores de una manera más sencilla. En una aplicación que implica el desarrollo de un algoritmo tan complejo como el que se acomete en este trabajo es conveniente facilitar la depuración del código.

- Favorece la escalabilidad del código. El desarrollo de este proyecto tiene una finalidad en primera instancia, completar la formación del alumno en el máster, pero desde un punto de vista a largo plazo es muy interesante pensar en futuras actualizaciones y mejoras, que serán mucho más fácil de implementar con el patrón utilizado.

- Facilita mucho el trabajo en equipo, ya que, mientras un grupo de trabajo se ocupa de las vistas (por ejemplo) otro puede encargarse del desarrollo de algoritmos y del tratamiento de datos. Como el alumno ha desarrollado el proyecto de manera individual no se ha podido explotar esta ventaja.

- El entorno de desarrollo usado en el proyecto facilita mucho la implementación de este patrón.

- Es un patrón muy conocido del que hay mucha información, y que el alumno ya conoce de desarrollos anteriores.

2.2.2. Desventajas:

- Para actividades de Android sencillas puede resultar un poco absurdo su uso. A lo largo del desarrollo ha habido actividades con una funcionalidad muy simple (como mostrar información y dar la opción de aceptar o cancelar) en las cuales la implementación del patrón ha resultado tediosa y no ha supuesto ninguna ventaja.

- Las primeras fases de desarrollo han sido más lentas.

- El número de clases desarrolladas se multiplica, por lo menos, por dos en los desarrollos de Android Studio.
  Las actividades siempre generan su vista de manera independiente, pero en el desarrollo usual se genera una clase Java que ejerce de controlador y modelo. Al dividir en ficheros independientes estas funcionalidades el tamaño del proyecto en cuanto a clases, por lo menos, se duplica.

- Se requiere el conocimiento de múltiples tecnologías. El alumno ha usado este patrón en más ocasiones, por tanto, esta desventaja no afectó.

Como se puede ver, las ventajas superan a las desventajas, en número e importancia, por tanto, teniendo en cuenta la manera en la que afectaría aplicar este patrón al proyecto a desarrollar, se ha decidido hacer uso de este.

3. Clases de diseño

Se mostrarán los paquetes que tiene cada una de las capas generadas por la aplicación del patrón Modelo-Vista-Controlador.

En cada paquete se podrán ver las clases y archivos que contiene.

3.1. Capa vista:

En el desarrollo de esta aplicación Android, como se ha mencionado anteriormente, la capa vista está formada principalmente por ficheros con extensión xml, que formarán la interfaz con la que se relacionará el usuario.

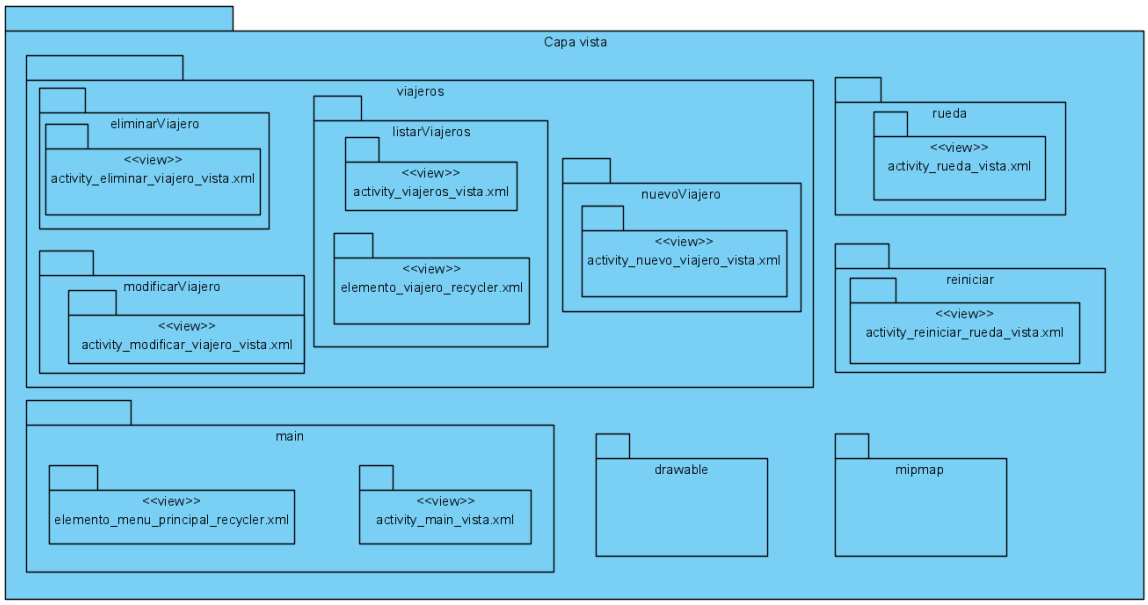

Ilustración 7: Diagrama capa vista

3.1.1. viajeros

Contiene todas las vistas de la gestión de viajero.

- eliminarViajero: Contiene las vistas que permiten al usuario eliminar al viajero que desee.
- listarViajeros: Contiene las vistas que permiten al usuario ver una lista con los viajeros almacenados en el sistema.
- nuevoViajero: Contiene las vistas que permiten al usuario registrar un nuevo viajero.
- modificarViajero: Contiene las vistas que permiten al usuario modificar al viajero que desee.

3.1.2. main

Contiene las vistas del menú principal.

3.1.3. rueda

Contiene las vistas de la gestión de horarios de rueda.

3.1.4. reiniciar

Contiene las vistas de la gestión de reinicio.

3.1.5. drawable

Contiene las imágenes y archivos que maquetan algunos elementos de la vista.

3.1.6. mipmap

Contiene los iconos de la aplicación que se mostrarán al usuario.

3.2. Capa controlador:

La capa controlador estará formada por los ficheros con extensión .java que ejerzan de intermediarios entre la capa vista y la capa modelo.

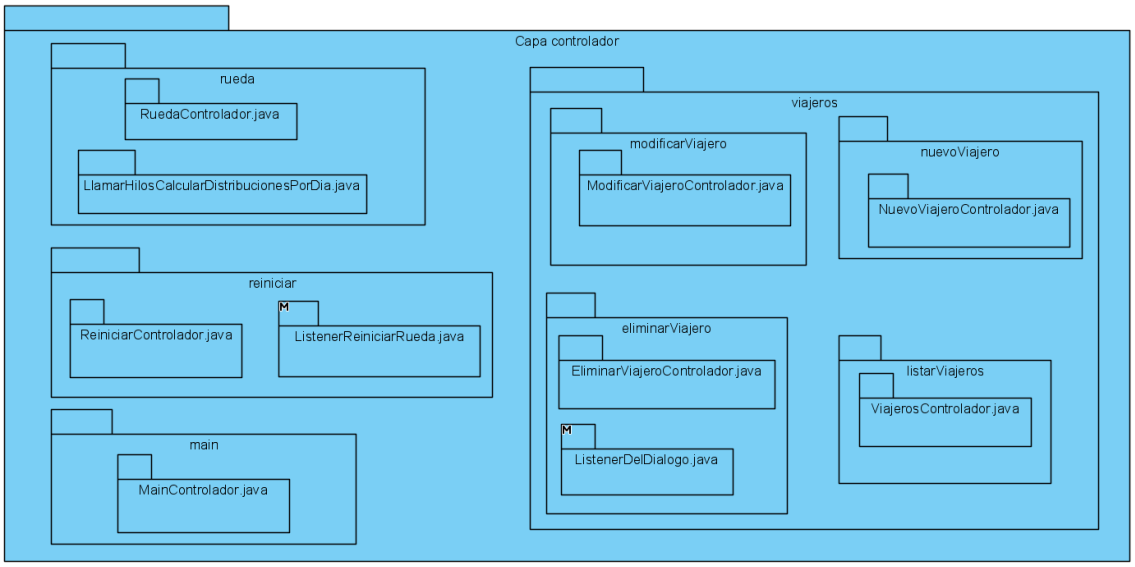

Ilustración 8: Diagrama capa controlador

3.2.1. viajeros

Contiene todos los controladores de la gestión de viajero.

- eliminarViajero: Contiene los controladores que permiten coordinar la eliminación de viajeros.
- listarViajeros: Contiene los controladores que permiten coordinar la visualización de los viajeros almacenados.
- nuevoViajero: Contiene los controladores que permiten coordinar el registro de un nuevo viajero.
- modificarViajero: Contiene los controladores que se permiten coordinar la modificación de los datos de un viajero seleccionado.

3.2.2. main

Contiene los controladores del menú principal.

3.2.3. rueda

Contiene los controladores de la gestión de horarios de rueda.

3.2.4. reiniciar

Contiene los controladores de la gestión de reinicio.

3.3. Capa modelo:

La capa modelo estará constituida por los ficheros con extensión.java que conformen la lógica de negocio y los datos que maneja el sistema.
Como el diagrama de la capa de modelo tiene un tamaño considerable, es complicado mostrarlo en una sola captura y que este sea legible, por tanto, se mostrará un vistazo general y se dividirá en dos imágenes que permitirán una mejor visualización.

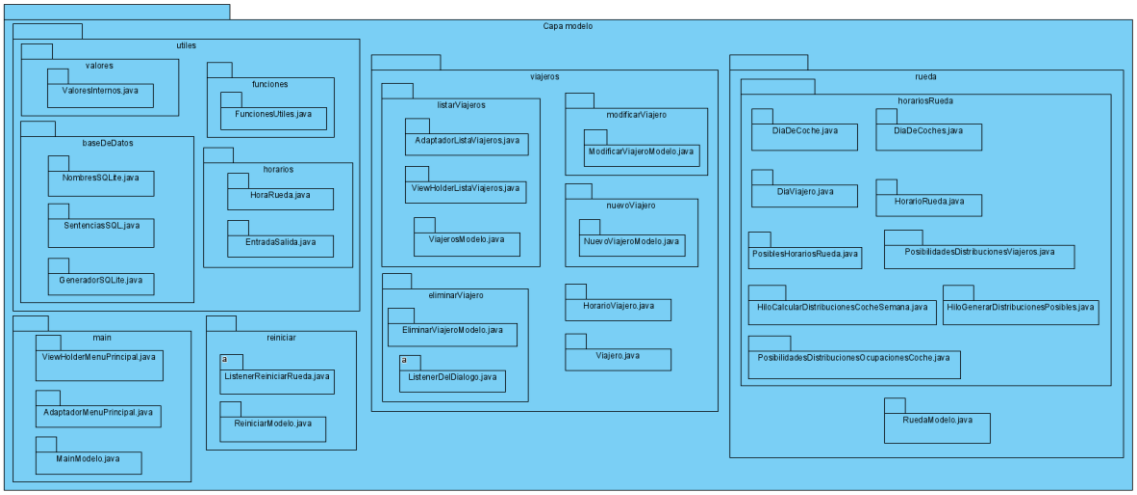

Ilustración 9: Diagrama capa modelo general

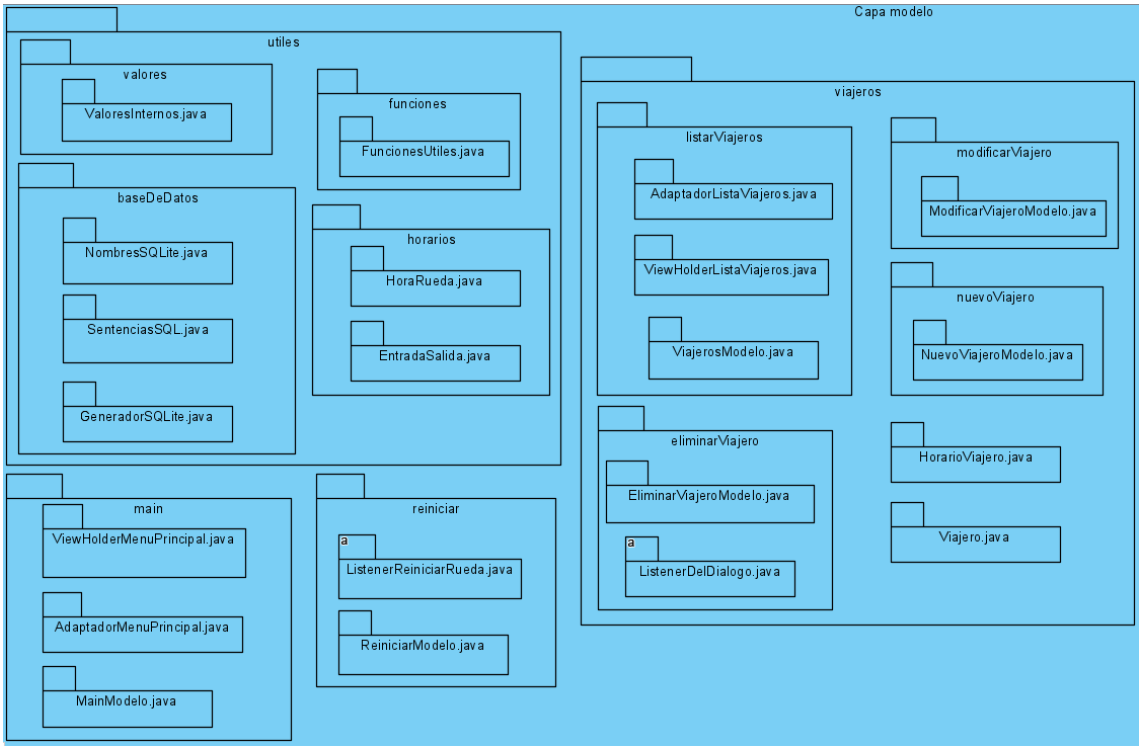

Ilustración 10: Diagrama capa modelo (parte 1)

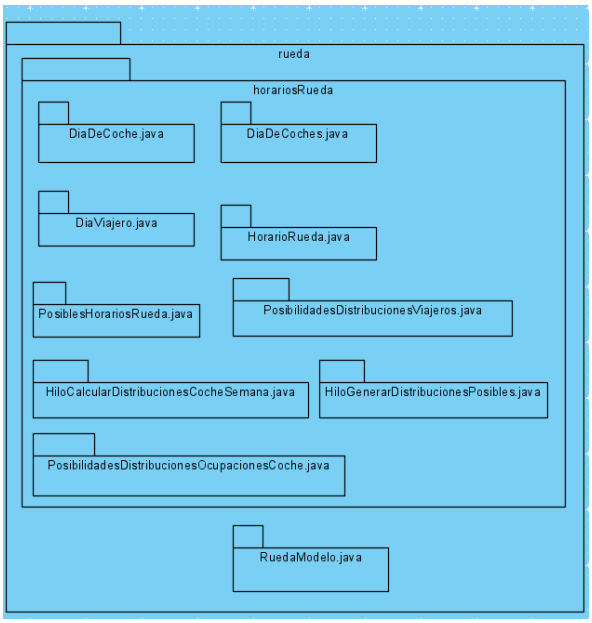

Ilustración 11: Diagrama capa modelo (parte 2)

3.3.1. viajeros

Contiene todos los modelos de la gestión de viajero.

- eliminarViajero: Contiene todos los datos y funciones que permiten eliminar un viajero del almacenamiento interno del sistema.

- listarViajeros: Contiene todos los datos y funciones que permiten visualizar una lista con los viajeros almacenados en el sistema.

- nuevoViajero: Contiene todos los datos y funciones que permiten añadir un nuevo viajero en el sistema.

- modificarViajero: Contiene todos los datos y funciones que permiten modificar los datos de un viajero en el sistema.

3.3.2. main

Contiene los datos y funciones que generan el funcionamiento del menú principal.

3.3.3. Rueda

Contiene los datos y funciones que permiten la gestión de los horarios de rueda.

- horariosRueda: Permite el almacenamiento de los datos correspondientes al horario de rueda y los métodos necesarios para calcular el horario de rueda óptimo y generar un PDF con los valores calculados.

3.3.4. reiniciar

Contiene los datos y métodos de la gestión de reinicio.

3.3.5. útiles

- baseDeDatos: Contiene todos los datos y métodos reutilizables que permiten generar la base de datos, gestionarla y operar con ella.

- funciones: Contiene funciones estáticas muy reutilizadas en el sistema.

- horarios: Contiene los datos y funciones sobre horarios, muy reutilizados en el sistema.

- valores: Contiene datos muy reutilizados en el sistema.

4. Vista Arquitectónica

Se procede a mostrar una vista arquitectónica global, marcada por el patrón Modelo-Vista-Controlador utilizado y en la que se hace referencia a las capas ya explicadas en el punto anterior.

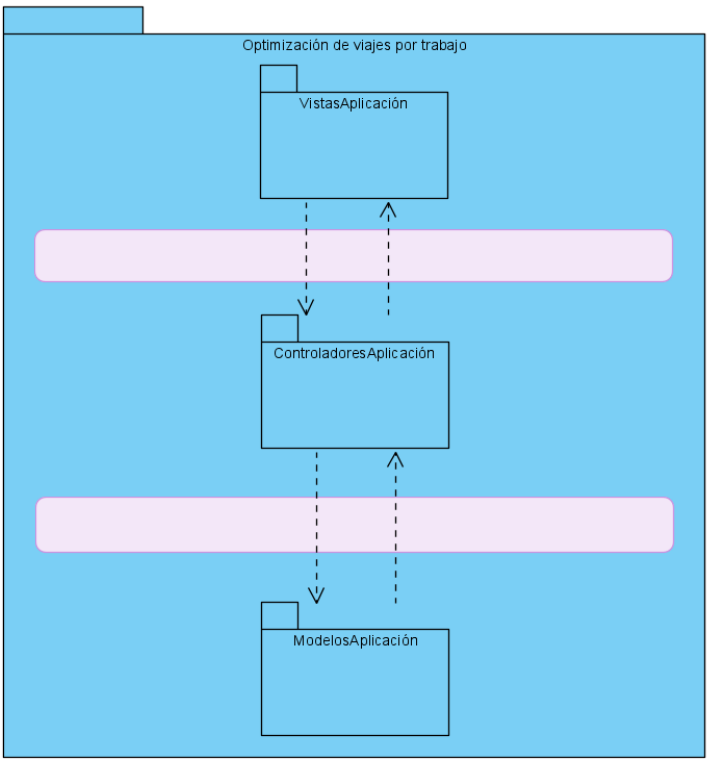

Ilustración 12: Diagrama vista arquitectónica

5. Realización de casos de uso

En este apartado se procede a refinar los diagramas de secuencia ya elaborados en el análisis del sistema del Anexo 3 indicando de una forma más precisa como se realizarán, utilizando ya nombres y elementos propios del desarrollo del sistema.

5.1. Diagramas de secuencia

Al igual que en el anexo anterior se harán diagramas de secuencia, solo que esta vez la abstracción será a un nivel más bajo.

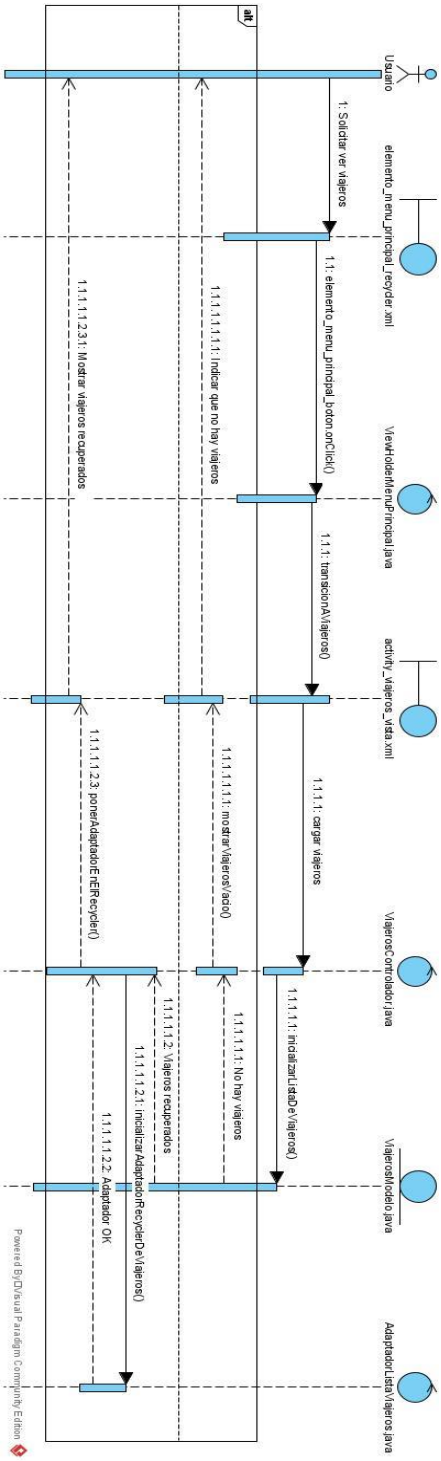

Ilustración 13: Diagrama de secuencia listar viajeros diseño

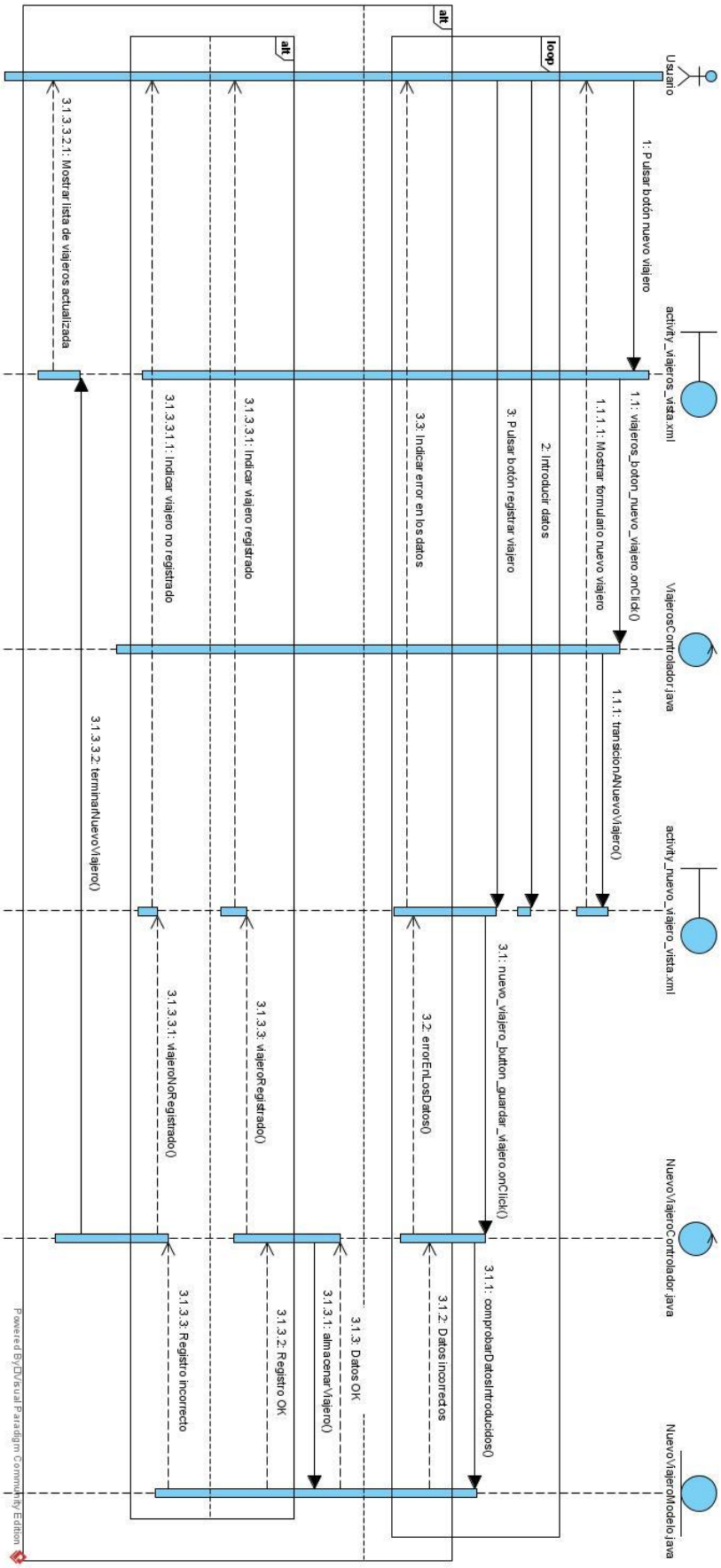

Ilustración 14: Diagrama de secuencia añadir viajero diseño

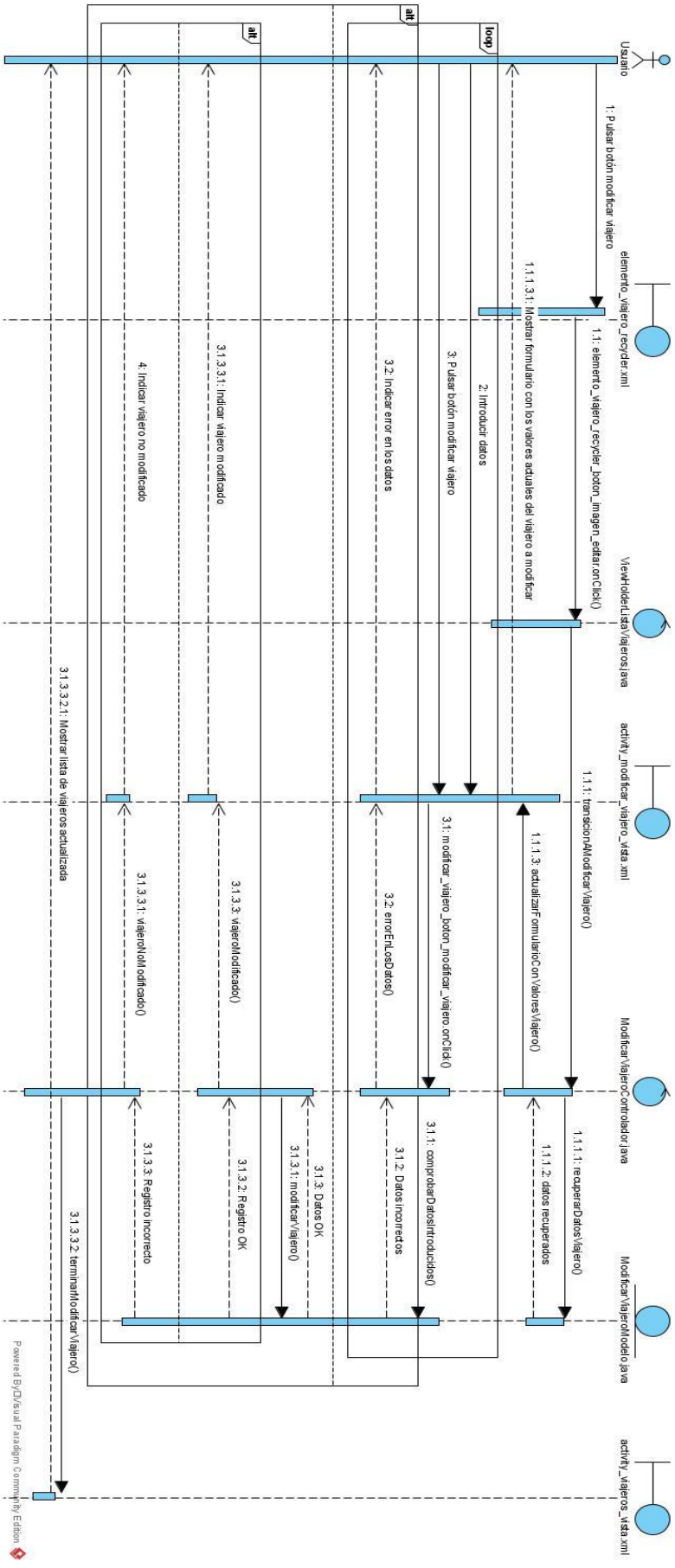

Ilustración 15: Diagrama de secuencia modificar viajero diseño

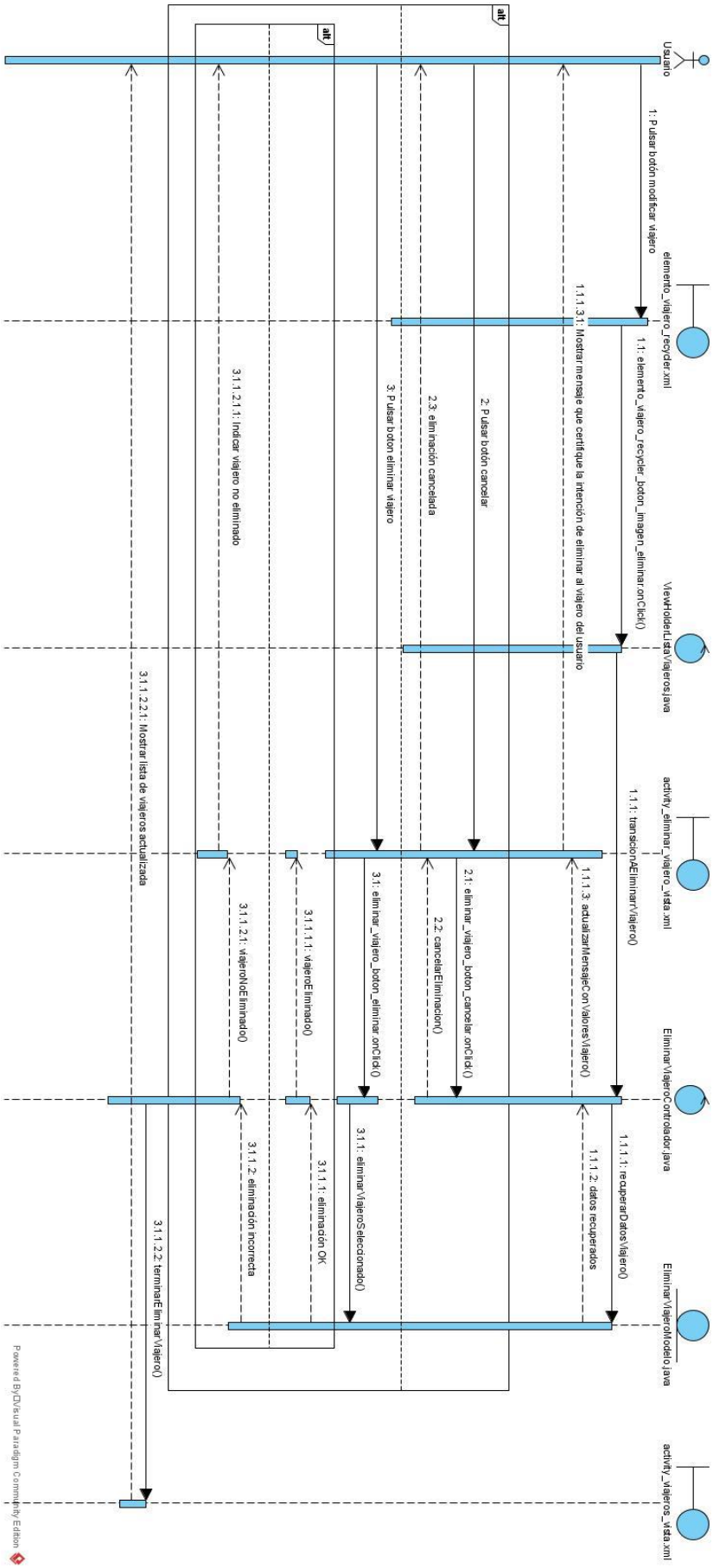

Ilustración 16: Diagrama de secuencia eliminar viajeros diseño

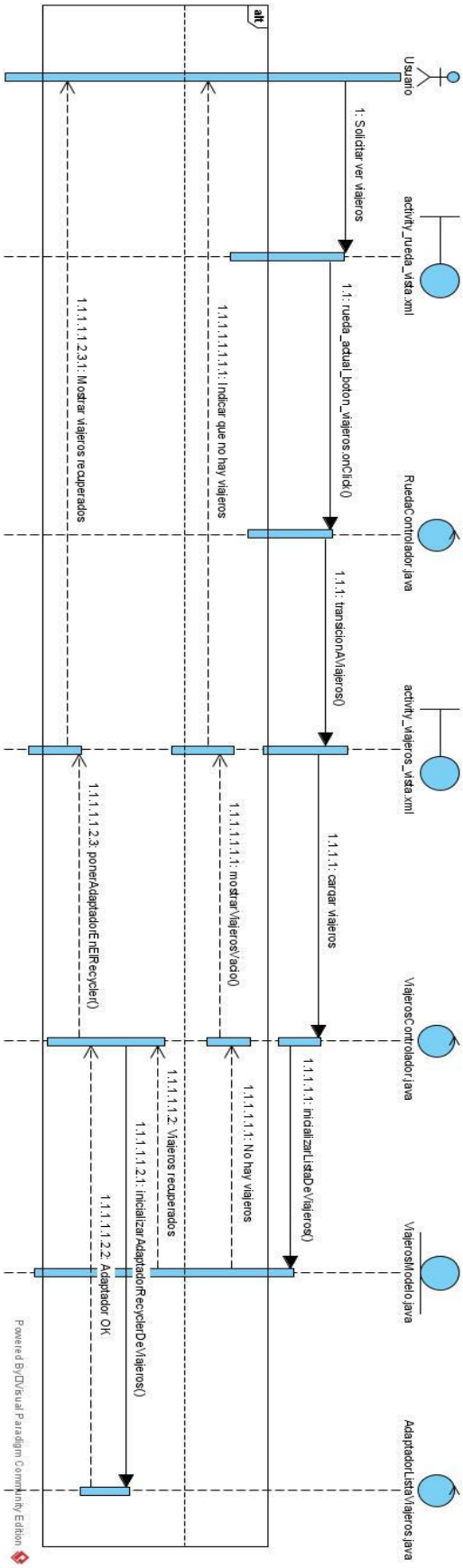

Ilustración 17: Diagrama de secuencia ir a viajeros diseño

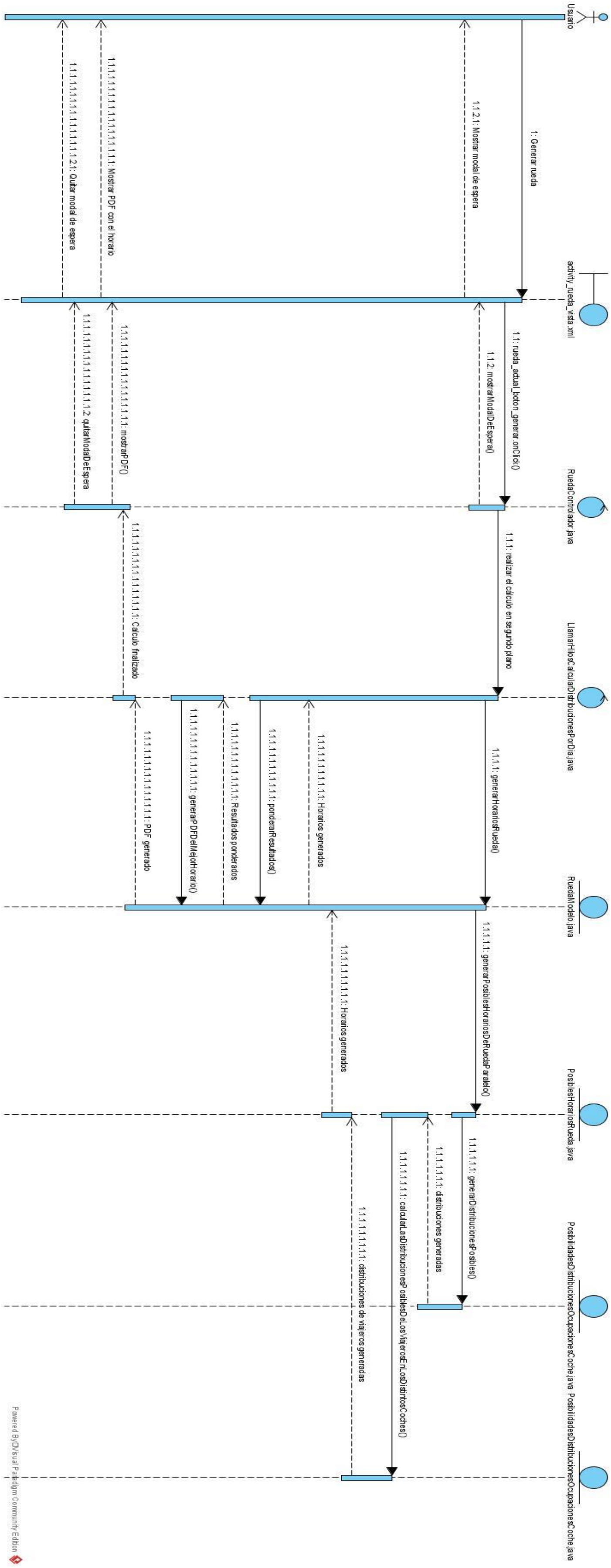

Ilustración 18: Diagrama de secuencia generar rueda diseño

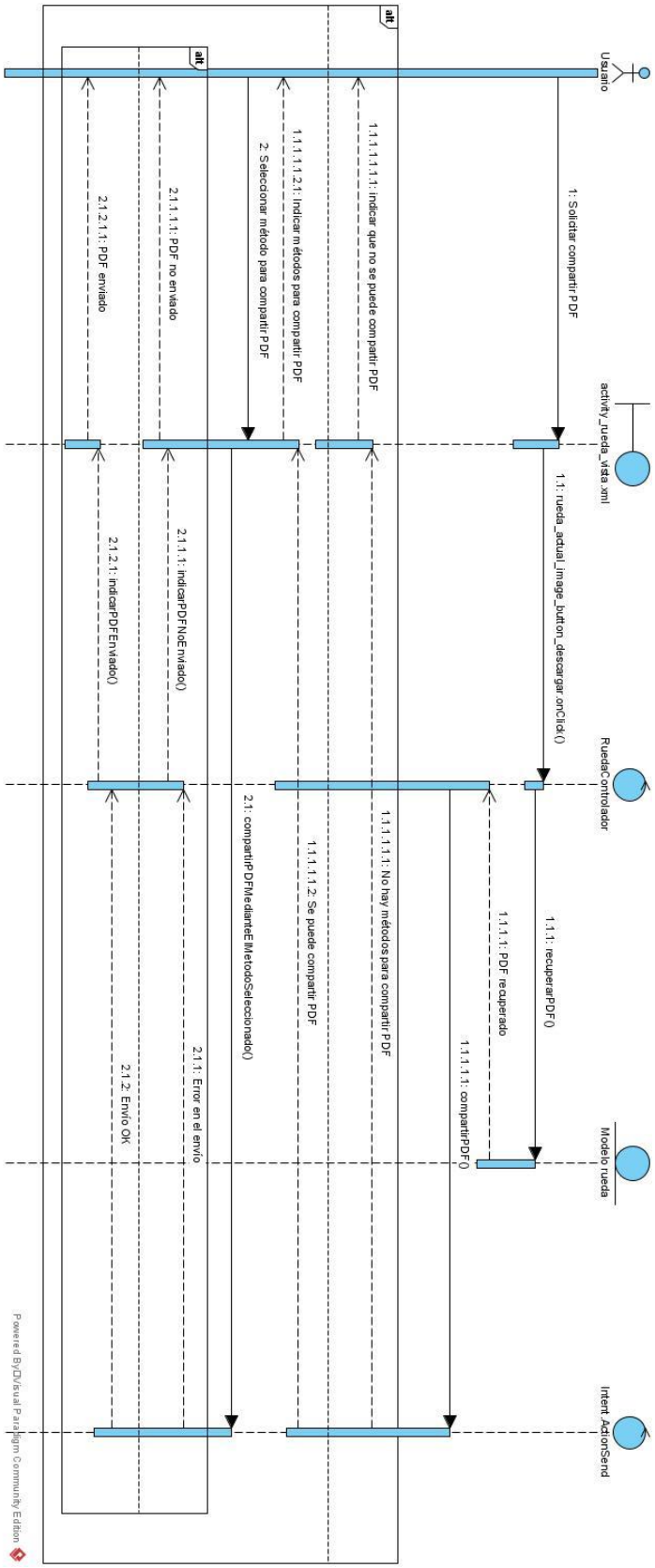

Ilustración 19: Diagrama de secuencia compartir PDF diseño

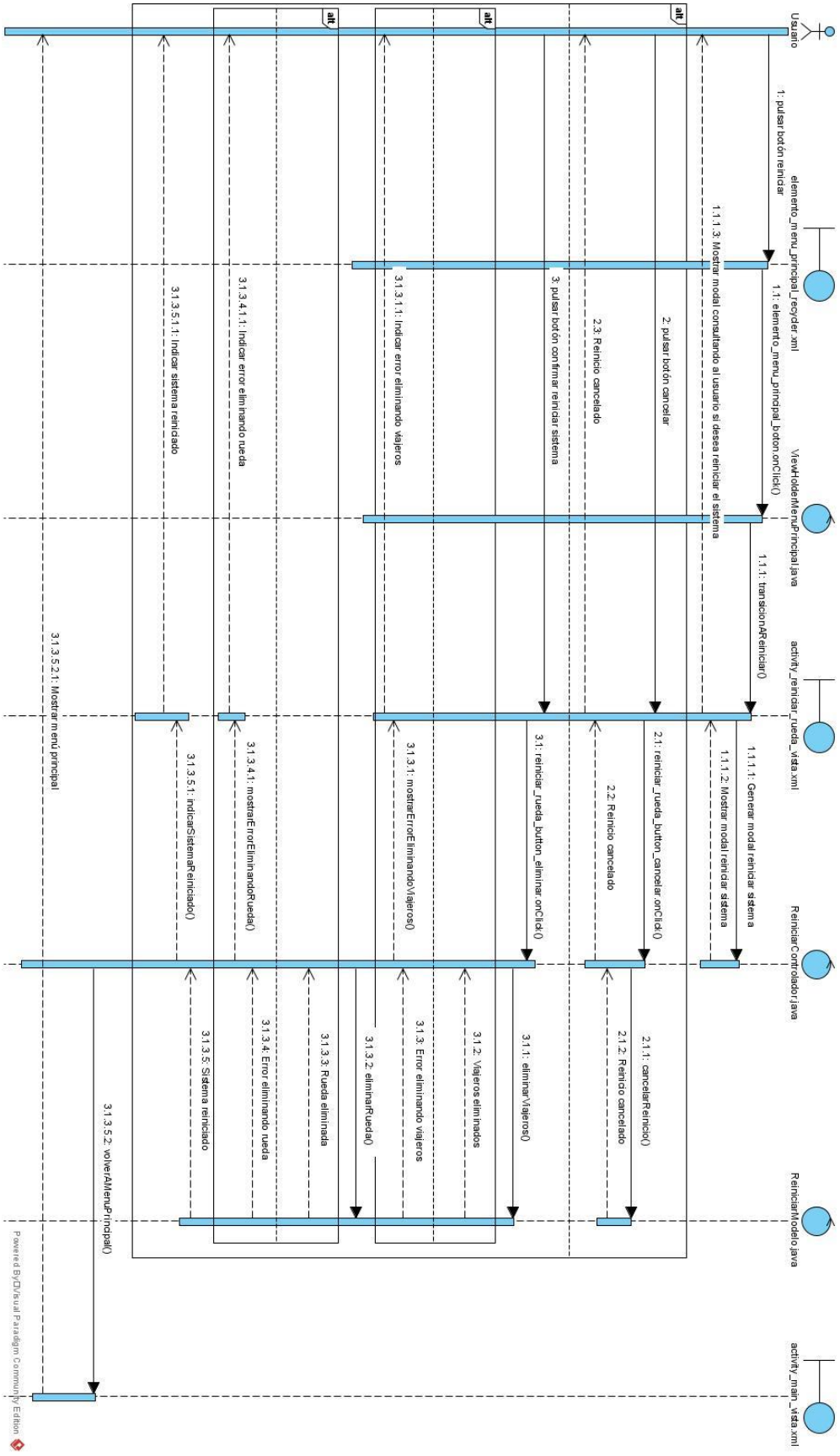

Ilustración 20: Diagrama de secuencia reiniciar sistema diseño

6. Modelo de despliegue

El objetivo de esta sección es representar gráficamente los elementos físicos que componen el sistema y la asignación de los componentes software en estos elementos. (manuel.cillero, s.f.)

Esta arquitectura representa tanto la arquitectura software como la hardware.

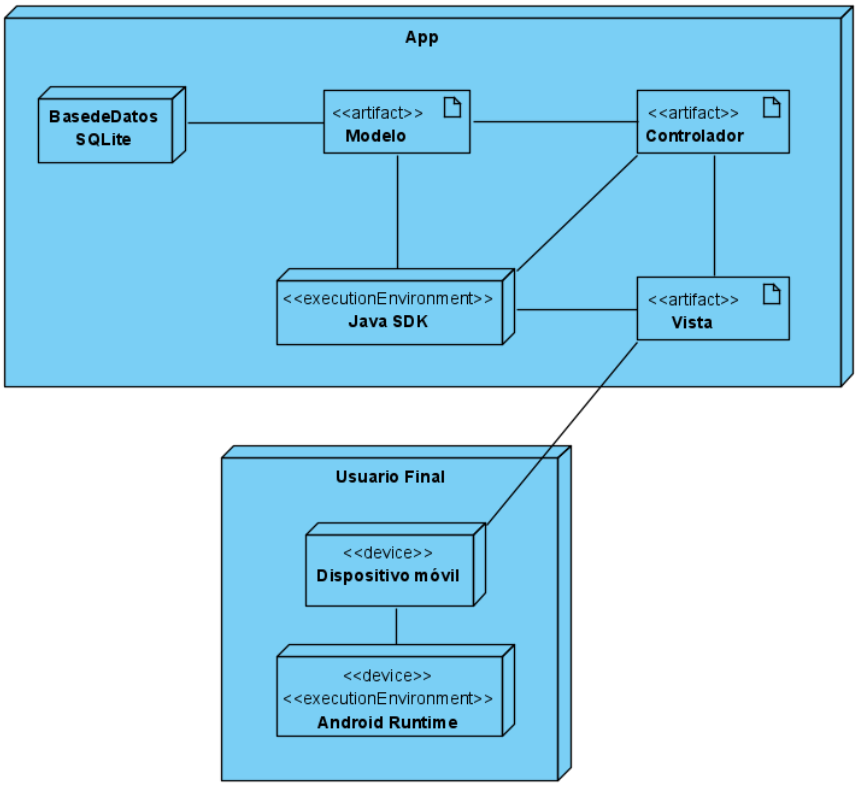

Ilustración 21: Diagrama de despliegue

La aplicación funciona en el smartphone gracias entorno de ejecución de Android.

El usuario interactúa con la vista de la aplicación, que se verá en el dispositivo móvil, y esta vista se relacionará con los artefactos de modelo y controlador de la misma manera que se ha explicado previamente, siendo este el comportamiento que se asocia inherentemente al patrón arquitectónico Modelo-Vista-Controlador.

Como no era necesario almacenar una gran cantidad de datos, se ha decidido utilizar una base de datos interna a la aplicación, esta funcionalidad la ofrece Android y permite un acceso más rápido y cómodo a la información salvaguardada.

7. Referencias

androidauthority. (s.f.). *https://www.androidauthority.com*. Obtenido de https://www.androidauthority.com/use-sqlite-store-data-app-599743/

campusmvp. (s.f.). *www.campusmvp.es*. Obtenido de https://www.campusmvp.es/recursos/post/que-es-el-patron-mvc-en-programacion-y-por-que-es-util.aspx

IBM. (s.f.). *https://www.ibm.com*. Obtenido de https://www.ibm.com/docs/es/rsm/7.5.0?topic=model-design

manuel.cillero. (s.f.). *https://manuel.cillero.es*. Obtenido de https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/diagrama-de-despliegue/

material.io. (s.f.). *https://material.io*. Obtenido de https://material.io/components/sliders/android