Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación
CC4901-01 - Práctica Profesional 1

Informe de Práctica Profesional I

NIC Chile Research Labs

Ivana Francisca Bachmann Espinoza
24 de febrero de 2016

Rut: 18.120.263-7
Carrera: Ingeniería Civil en Computación
Correo: ivana.bachmann@gmail.com
Celular: (+56) 9 6591103

Índice general

Capítulo 1

Resumen

El trabajo fue realizado en el laboratorio de investigación NIC Chile Research Labs . Este consistió en el desarrollo de extensiones gráficas para la aplicación para dispositivos móviles *Adkintun Mobile*.

Adkintun Mobile tiene como objetivo permitirle al usuario mediante diferentes métricas chequear la calidad de la conexión a Internet móvil de su dispositivo, ya sea en cuanto a la calidad de conexión en un área determinada, tipos de señal al que se ha mantenido conectado o tiempos sin conexión, además puede mantener una evaluación de la calidad de la conexión desde la perspectiva del usuario.

Las extensiones gráficas realizadas tienen la meta de mostrar de forma clara y precisa datos recolectados por dicha aplicación de métricas existentes, que antes eran posibles de visualizar únicamente de forma diaria, en un espacio de tiempo de aproximadamente 30 días. Dichas extensiones se realizan con el objetivo de mejorar la experiencia de usuario, para que este pueda entender mejor los datos que recolecta la aplicación *Adkintun Mobile* y además este pueda comprender de forma más completa el servicio de conexión móvil que su compañía le está ofreciendo. Estas extensiones fueron realizada en dos etapas, la etapa de refactoring y la etapa de implementación del diseño, siendo esencial la etapa de refactoring dadas las condiciones previas de las partes a extender.

El trabajo realizado se encuentra actualmente integrado a la aplicación oficial de *Adkintun Mobile* y se encuentra disponible para descarga en dispositivos *Androrid*.

En cuanto a los aprendizajes obtenidos en el ámbito técnico se destaca el conocimiento adquirido en programación de aplicaciones para Android, en el uso de XML para desarrollo de aplicaciones para Android y el uso de git. En cuanto a otras habilidades aprendidas se destaca el trabajo en equipo con un equipo multidisciplinario.

Capítulo 2

Introducción

2.1. Lugar de trabajo

La práctica profesional serealizó en NIC Chile Research Labs[1], laboratorio de investigación aplicada en las áreas de redes computacionales, sistemas distribuidos y protocolos de Internet. Se encuentra ubicado en Av. Almirante Blanco Encalada #1975, Santiago, Chile. El trabajo se realizó de forma presencial en las instalaciones del laboratorio, entre el 15 de diciembre de 2014 y el 30 de enero de 2015.

2.2. Grupo de trabajo

En NIC Chile Research Labs trabajan investigadores de tiempo completo, desarrolladores experimentados, profesionales de otras áreas y estudiantes tanto de pregrado como postgrado. En particular en la oficina que se realizó el trabajo fue usada de forma regular por otras 6 personas de las cuales 3 se encontraban en el mismo equipo de la practicante. En cuanto al proyecto descrito en el presente informe el equipo de trabajo fue conformado por un desarrollador e investigador, Felipe Lalanne, el cual fue el encargado del proyecto, una diseñadora y otro practicante, todos ellos presentes en la oficina.

2.3. Equipos y Software

2.3.1. Software

El software utilizado para desarrollar el trabajo fue Java como lenguaje de programación (a través del IDE Eclipse usando Android Development Tools), XML como lenguaje de marcado, git para manejo de control de versiones, el sistema operativo Android y el sistema operativo Ubuntu. Además cabe destacar a la aplicación para android *Adkintun Mobile* en la cual se trabajó. A continuación se presenta una breve descripción los elementos más relevantes para el desarrollo y comprensión del trabajo realizado:

1. **Java:** Java es un lenguaje de programación de propósito general, orientado a objetos, diseñado para tener tan pocas dependencias de implementación como sea posible. En el contexto del trabajo realizado, Java a través de Eclipse usando las ADT (Android Development Tools) sirvió como lenguaje de programación para aplicaciones en Android [2].

   Un recurso importante en el desarrollo de aplicaciones para Android en Java, y de importancia para la comprensión del presente informe, corresponde a las *actividades*, las cuales son todas aquellas clases que heredan directamente de la clase *Activity* o bien heredan de una clase que es subclase de *Activity*. En las actividades es donde se establece qué se mostrará al usuario en una de las pantallas de la aplicación, así como su comportamiento al ser creada, pausada, o destruida. Una aplicación para Android consta de una o más actividades que interactúan con el usuario.

2. **XML:** XML corresponde a un lenguaje de marcado, es decir permite establecer reglas de formato que sean comprensibles tanto para quien las establece como para la máquina. En el caso particular de la práctica XML fue utilizado para establecer las reglas de formato en cuanto al diseño de interfaces para las distintas *actividades* (formato de *Layouts, Tabs, ProgressBars*, etc).

3. **Adkintun Mobile:** *Adkintun Mobile* es una aplicación para dispositivos Android que nace como proyecto de investigación en NIC Chile Research Labs cuyo objetivo es establecer un sistema de monitoreo y medición de la calidad del servicio de Internet móvil en el territorio nacional. Para ello la aplicación monitorea y almacena información sobre uso de señal, ya sea wi-fi, móvil o sin conexión, tipo de señal móvil disponible, cantidad de bytes enviados y recibidos, y calidad de la señal recibida por el dispositivo desde antena. La información obtenida es accesible por los usuarios de la aplicación a través de diferentes gráficos, permitiendoles monitorear la calidad de servicio que reciben por parte de su compañía telefónica.

   El trabajo realizado corresponde a modificaciones realizadas dentro de esta aplicación con el objetivo de llegar a más usuarios ofreciendoles mejoras en las funcionalidades.

2.3.2. Equipo

En cuanto a los equipos computacionales, el laboratorio provee tanto equipos estacionarios como portátiles para aqullos que se encuentran trabajando o realizando prácticas. Para el proyecto desarrollado en particular se utilizó un equipo estacionario el cual fue provisto por el laboratorio. También se utilizaron diversos celulares con sistema operativo Android los cuales se facilitan por el laboratorio para probar aplicaciones, además del celular propio de la practicante.

2.4. Situación Previa

A continuación se describirá la situación previa al trabajo de la practicante. Esta descripción estará centrada en las interfaces gráficas que provee *Adkintun Mobile* para la visualización de los datos obtenidos del dispositivo móvil a través de la aplicación pues son las carácteristicas relevantes para la comprensión del trabajo realizado. Previo al trabajo realizado la aplicación *Adkintun Mobile* tenía un conjunto de visualizaciones de datos recogidos por la aplicación, los cuales eran:

1. **Tiempo conectado:** Muestra la cantidad de tiempo sin conexión, contectado a Internet móvil y conectado a wi-fi. La información es mostrada en intervalos de tiempo asignados a cada caso, estos intervalos son mostrados desde las 00:00 hrs del mismo día en que es consultada la información. (Ver figura 4.1)
2. **Tipo de señal recibida:** Muestra la cantidad de tiempo que estuvieron disponibles los distintos tipos de señal móvil (3G, H, H+, etc). Al igual que en el punto anterior la información es mostrada en intervalos de tiempo asignados según el tipo de señal disponible durante dicho intervalo y los intervalos comienzan a las 00:00 hrs del mismo día en que se consulta la información. (Ver figura 4.2)
3. **Tráfico móvil por aplicación:** Muestra el tráfico móvil por aplicación en bytes. Muestra la información de las últimas 2 horas y de los últimos 30 días. (Ver figura 4.3)
4. **Evalúa tu conexión:** El usuario puede darle hasta 5 estrellas a su conexión móvil, para evaluar la satisfación del usuario con su conexión. (Ver figura 4.4)
5. **Mapa de calidad de señal:** Muestra un mapa con los puntos donde se encuentran las antenas que ha utilizado el dispositivo móvil para conectarse a la señal móvil, asociado a un color que muestra la intensidad de la señal percibida por el dispositivo. (Ver figura 4.5)

2.5. Descripción general del trabajo realizado

El objetivo del trabajo desarrollado fue el de proveer nuevas interfaces gráficas a los usuarios de *Adkintun Mobile* de forma que este pudiese interpretar de mejor forma la información recolectada

por la aplicación. Para ello se agregaron nuevas visualizaciones a "Tiempo conectado" y "Tipo de señal recibida" de forma que ahora mostrasen además la información recolectada a lo largo de los últimos 30 días, pues como se describió anteriormente estas interfaces mostraban la información únicamente de forma diaria.

Para llevar a cabo el trabajo este fue dividido por la practicante en tres etapas que se describen brevemente a continuación:

1. **Exploración del código previo:** Dado que el trabajo consistió en agregar interfaces de usuario a una aplicación ya existente, fue necesario comenza el trabajo explorando el código existente de forma de identificar aquellas clases y actividades involucradas en las partes a extender.

2. **Refactoring:** Dadas las condiciones previas en las que se encontraban las clases que implementaban las interfaces gráficas diarias fue necesario realizar un *refactoring*, creando nuevas clases asociadas a las clases previamente encargadas de las interfaces, de forma de separar las responsabilidades.

3. **Implementación de las interfaces de usuario:** La segunda etapa correspondió a la implementación de las interfaces mismas, es decir la implementación de los diseños gráficos con los cuales el usuario iba a relacionarse finalmente.

Capítulo 3

Trabajo realizado

Como se ha mencionado previamente el trabajo consistió en agregar una visualización de los datos de los últimos 30 días a "Tiempo conectado" y "Tipo de señal recibida" las cuales ya mostraban datos diarios. Esto pues si bien dichas visualizaciones son útiles no permitian al usuario entender a cabalidad el comportamiento de su señal móvil. Por ejemplo podía ocurrir que un día el item"Tipo de señal recibida" se tuviese que el 90 % de ese día se tuvo conexión a señal 3G, sin embargo podría ser que dicho día fuese una excepción y que en general, en un periodo de 30 días, sólo un 60 % del tiempo se tuviese conexión a 3G. El formato pedido para añadir dicha funcionalidad fue el de pestañas, es decir tener una pestaña que permita mostrar la información diaria y una pestaña que permita acceder a la información acomulada de los últimos 30 días. Teniendo en cuenta la necesidad de agregar más opciones de visualización de datos y el formato pedido para esta, se procedió a desarrollar el trabajo en las etapas que describiremos a continuación.

3.1. Etapa 0: Análisis del código previo

El primer paso para realizar las modificaciones solicitadas fue identificar las clases involucradas en las vistas de "Tiempo conectado" y "Tipo de señal recibida". Esta exploración se realizó al mismo tiempo que la practicante aprendió sobre los elementos específicos para el desarrollo de aplicaciones para Android, como lo son las actividades, fragmentos y utilización de los elementos para construir la UI.

3.1.1. Clases: "Tiempo conectado" y "Tipo de señal recibida"

Por medio de un análisis exploratorio del código de la aplicación se encontró que las actividades encargadas dichas secciones eran:

1. **ConnectedTimeActivity** que se encarga de "Tiempo conectado"
2. **ConnectionTypeActivity** encargada de "Tipo de señal recibida"

Se encontró que cada clase se encargaba de la extracción de los datos a mostrar y de generar la visualización de los mismos para ser mostrados en la UI por lo que no se tenía la participación de más clases. De forma más detallada cada actividad poseía las siguientes partes:

1. **Extracción de datos:** Cada actividad se comunicaba con la base de datos a través de *Sugar* para extraer los datos necesarios. *Sugar* [3] es un framework de ORM (Object-relational mapping), es decir otorga funcionalidades para acceder a la base de datos a través de objetos. A medida que se extraía información esta era añadida a estructuras de datos que servirían para crear los elementos de la UI.
2. **Generación de la vista:** La generación de la vista se encargaba tanto de ubicar en pantalla los elementos estáticos de la vista (simbologías, fondos, fonts, etc) como aquellos elementos que representarían la información recogida. Los elementos estáticos de la vista eran mostrados y cargados al mismo tiempo que se realizaba la extracción de datos utilizando *threads*. Una vez la extracción de datos se completaba entonces se procedía a mostrar los elementos gráficos que representaban cada conjunto de datos en cada clase.

Cabe destacar que tanto ConnectedTimeActivity como ConnectionTypeActivity realizaban los mismos pasos para ejecutarse, cambiando únicamente en el tipo de datos extraidos. En particular ambos mostraban el mismo tipo de gráfico para representar sus datos.

3.1.2. Pestañas:

Dentro del conjunto de visualizaciones de datos se encuentra "Tráfico móvil por aplicación", la cual muestra mediante pestañas datos diarios y de los últimos 30 días. Dado esto último se decidió continuar con el análisis exploratorio para observar el funcionamiento de las pestañas en dicha visualización. Luego de revisar el código se encontró que la implementación de las pestañas, y el cómo se mostraba la información, se encontraba en la clase AplicationTrafficActivity. En esta clase se observaron las siguientes etapas para implementar las pestañas:

1. **Inicialización:** Primero es necesario obtener un objeto de tipo TabHost el cual contendrá las pestañas, el cual se obtiene desde el XML asociado a la actividad un objeto TabHost de la siguiente forma:

```
1 TabHost tabHost = (TabHost) findViewById(R.id.tabHost);
```

   Luego este se prepara para que le sean añadidas nuevas pestañas mediante

```
tabHost.setup();
```

2. **Adición de pestañas:** Una vez se ha inicializado el TabHost se procede a configurar y añadir las pestañas. La pestañas corresponden a objetos de tipo TabSpec el cual se inicializa como

```
TabSpec specDay = tabHost.newTabSpec(DAY_TAB_SPEC);
```

   Donde DAY_TAB_SPEC corresponde a un *tag* (string de identificación).

   Además cada TabSpec contiene la frase que se mostrará en la pestaña (ej. día o últimos 30 días) y su configuración visual dada por un XML (color al estar o no presionado, tamaño y posición de texto, etc).

   Una vez terminada la inicialización y configuración de cada TabSpec estos son añadidos al TabHost mediante

```
tabHost.addTab(specDay);
```

3. **Configuración de comportamiento:** Finalmente se procede a definir el comportamiento de cada pestaña mediante un Listener. En el caso en que una de las pestañas muestra un día y la otra muestra los últimos 30 días esto se ve como

```
tabHost.setOnTabChangedListener(new OnTabChangeListener() {
                    @Override
                    public void onTabChanged(String tabId) {
                            /* ... */
                            boolean showingDay = !tabId.equals(MONTH_TAB_SPEC);
                            if (showingDay) {
                                    /* comportamiento diario*/
                            } else {
                                    /* comportamiento de los ultimos 30 dias */
                            }
                            /* ... */
                    }
            });
```

3.2. Etapa 1: Refactoring

Una vez encontradas las clases involucradas en las áreas a modificar, entendido su funcionamiento y el de la implementación de pestañas como Tabs, se decidió realizar un refactoring. El objetivo de este refactoring fue el de permitir la extensibilidad del código mediante la separación de responsabilidades en diferentes clases. De esta forma añadir una pestaña con información diferente correspondería a agregar un par de clases nuevas y no un cambio mayor en el código de la actividad.

3.2.1. Diseño

Para llevar a cabo el refactoring el primer paso es determinar qué patrón de diseño se va a utilizar. Como se señaló anteriormente el proceso de construcción de la vista consta de una parte de extracción de datos, configuración en elementos para la UI y muestra de los mismos en pantalla, por lo que se decidió utilizar el patrón de diseño *Modelo Vista Controlador* (MVC).

El MVC consta de las siguientes partes:

1. **Modelo:** Su responsabilidad es el de manejar la representación de la información con la cual se está trabajando, por lo tanto le corresponde interactuar y gestionar dicha información.
2. **Controlador:** Es el intermediario entre la vista y el controlador, le corresponde responder a eventos y enviar peticiones al modelo.
3. **Vista:** Muestra la información de forma gráfica al usuario, en un formato apropiado. La información mostrada corresponde a la información que provee el modelo.

Aplicando este modelo a las responsabilidades identificadas en las actividades se obtuvo lo que cada parte del MVC debe realizar

- El **Modelo** debe encargarse de la extracción de datos, pues es el modelo el que debe encargarse de interactuar y gestionar la información.
- La **Vista** debe encargarse sólo de mostrar los datos y lanzar eventos, luego la vista debe ser la actividad misma (ConnectedTimeActivity o ConnectionTypeActivity).
- El **Controlador** debe ser quien actúe como puente entre los elementos anteriores, por ello el controlador se encargará de generar a partir del modelo los elementos de la vista que representarán dicha información para luego entregarselos a la vista y que esta los muestre en pantalla. Además será el controlador quien interactúe con el modelo cada vez que ocurra un evento, en este caso, cada vez que se cambie la pestaña.

3.2.2. Clases

Una vez resuelto el patrón de diseño a implementar se procedió a crear clases que pudiesen definir el comportamiento esperado del modelo y del controlador. Estas clases fueron **GraphicStatisticsBuilder** y **StatisticInformation**.

La clase **GraphicStatisticsBuilder** es un *interface de java* y su objetivo es definir el comportamiento mínimo que debe tener un controlador. El método createGraphicStatistic recibe un controlador de tipo StatisticInformation e interactuando con él construye un objeto View que representa

de forma gráfica la información obtenida desde el objeto StatisticInformation.

```
public interface GraphicStatisticsBuilder {
	public View createGraphicStatistic(StatisticInformation a);
}
```

Por otro lado **StatisticInformation** corresponde a una *clase abstracta* que representa al modelo. Su objetivo es ser extendida por todas aquellas clases que representen algún modelo de datos. Esta clase provee los métodos mínimos que se necesitarán implementar para interactuar con el controlador, además de permitirle al controlador referirse a cualquier modelo a través de un objeto de tipo StatisticInformation.

```
public abstract class StatisticInformation {
	ArrayList<Object> information;
	public StatisticInformation(){
		information=null;
	}
	public abstract void setStatisticsInformation();
	public abstract ArrayList<Integer> getColors();
	public abstract ArrayList<Float> getValues();
	public abstract ArrayAdapter<Integer> getAdapter();
}
```

El método setStatisticsInformation es el encargado de extraer la información de la base de datos y almacenarla en el formato adecuado. Los métodos getters aparecen como una forma de permitirle al controlador obtener aquellos datos que eran utilizados con mayor frecuencia dadas las vistas que se tenían y las vistas que se deseaban agregar.

A partir de estas dos clases se crearon todas las clases necesarias para realizar el refactoring y la extensión gráfica pedida. A continuación se describirán las implementaciones y subclases de las clases recién descritas. (ver figuras 4.6 y 4.7)

Clases que implementan a *GraphicStatisticsBuilder*:

a) **DoughnutChartBuilder:** Su objetivo es generar un gráfico de dona a partir de la información obtenida desde el modelo. Los gráficos de dona se contruyen a partir de un objeto DoughnutChart, estos ya eran utilizados para mostrar la información tanto en "Tiempo conectado" como en "Tipo de señal recibida". Luego este controlador es utilizado para mostrar la información en las pestañas diarias de cada visualización.

b) **FatBarBuilder:** En este caso el controlador genera un objeto de tipo FatBar para graficar la información obtenida desde el modelo. Los gráficos tipo FatBar fueron desarrollados por la practicante y serán abordados en detalle en la sección de *Implementación de interfaces de usuario.*

c) **ListBarBuilder:** El controlador pretende generar un ListView que muestra una FatBar en el segmento inicial, y la simbología en los demás segmentos. Cabe mencionar que un

ListView es una vista que muestra una lista de elementos y que puede ser configurada fila por fila para mostrar elementos específicos.

Clases que extienden a *StatisticInformation*:

a) **DailyConnectedTimeInformation:** Este modelo entrega los datos para generar la vista diaria de la visualización "Tiempo conectado". La extracción de datos propiamente corresponde a aquella que se encontró en la actividad original *ConnectedTimeActivity*.

b) **DailyConnectionTypeInformation:** Como en el caso anterior este modelo entrega los datos para generar la vista diaria de "Tipo de señal recibida", y la forma de extraer los datos corresponde a la encontrada en la versión original de *ConnectionTypeActivity*.

c) **MonthlyConnectionTypeInformation:** En este caso el modelo entrega los datos necesarios para generar la vista de los últimos 30 días de información para la actividad *ConnectionTypeActivity*. Esta actividad no poseía una forma de extraer los datos previa. La practicante se encargó de proveer la clase MonthlyConnectedTypeActivity, implementar los métodos correspondientes de la clase y de almacenar los datos en la estructura correcta para ser entregados al controlador, sin embargo el cómo extraer los datos fue un trabajo realizado por el otro practicante del equipo.

d) **MonthlyConnectedTimeInformation:** Esta clase se encarga de proveer el modelo para la actividad ConnectedTimeActivity. El trabajo consistió en proveer la clase que contiene al modelo, sus métodos y la correcta forma de almacenar los datos para entregarlos al controlador, aún así la extracción misma de los datos fue implementada por el otro practicante del equipo.

3.2.3. Funcionamiento

Ahora que se conocen las clases necesarias para implementar el sistema de pestañas para las visualizaciones "Tiempo conectado" y "Tipo de señal recibida" se explicará la utilización de las mismas para la construcción de las pestañas. Para llevar a cabo la explicación tomaremos como ejemplo las llamadas al controlador desde la actividad encargada de "Tiempo conectado": ConnectedTimeActivity. El uso en el caso de ConnectionTypeActivity es análogo.

Pasos

1.- Primero se genera un elemento de la clase StatisticInformation genérico al cual llamamos *statistic* el cual se inicializa con una de sus subclases según se esté en la pestaña que debe mostrar la vista diaria o en la pestaña que muestra la información de los últimos 30 días. En particular notemos que en el primer acceso a la visualización de "Tiempo conectado" se debe mostrar alguna pestaña, en este caso se parte mostrando la información diaria.

```
statistics = new DailyConnectedTimeInformation(context);
tabHost.setOnTabChangedListener(new OnTabChangeListener() {
    @Override
    public void onTabChanged(String tabId) {
        /* showingDay: true if we are on the DAY TAB */
        showingDay = !tabId.equals(MONTH_TAB_SPEC);
        if (showingDay) {
            statistics = new DailyConnectedTimeInformation(context);
        } else {
            statistics = new MonthlyConnectedTimeInformation(context);
        }
        loadGraphics(statistics);
    }
});
loadGraphics(statistics);
```

2.- Luego de seleccionar el modelo correspondiente este es entregado al método loadGraphics el cual recibe un objeto de tipo StatisticInformation. En este método se cargan los elementos gráficos estáticos de cada pestaña y se cargan los gráficos que correspondan según si el tipo del modelo es mensual o diario, esto se hace seleccionando el controlador correspondiente.

```
private synchronized void loadGraphics(final StatisticInformation statistic) {
    /* Carga de elementos estaticos de la vista
     * segun si esta es diaria o no.
     */
    (new Thread() {
        @Override
        public void run() {
        /* ... */

        /* ... */
        if (showingDay) {
            DoughnutChartBuilder chartBuilder = new DoughnutChartBuilder(chartAux, chartDiameter);
            chartAux = (DoughnutChart) chartBuilder.createGraphicStatistic(statistic);
            setChartVisible(chartAux);
        } else {
            ListBarBuilder barBuilder = new ListBarBuilder(listView, emptyView);
            final ListView barView = (ListView) barBuilder.createGraphicStatistic(statistic);
            listView = barView;
        }
        /* ... */
            }
    }).start();
}
```

3.- El controlador se encarga de extraer los datos desde el modelo y entregarle una vista hecha al modelo. A continuación se muestra el caso particular de DoughnutChartBuilder.

```
public class DoughnutChartBuilder implements GraphicStatisticsBuilder {
    private DoughnutChart chart;
    private float chartDiameter;
    public DoughnutChartBuilder(DoughnutChart chart, float chartDiameter){/* ... */}
    @Override
    public View createGraphicStatistic(StatisticInformation info) {
        info.setStatisticsInformation();
        final ArrayList<Integer> colors = info.getColors();
        final ArrayList<Float> values = info.getValues();
        /* se configuran las opciones del grafico de dona */
        chart.setOffset(88f);
        chart.setDiameter(chartDiameter);
        chart.setColors(colors);
        chart.setValues(values);
        return chart;
    }
}
```

4.- Finalmente la vista recibe un objeto View desde el controlador que puede mostrar directamente en la pantalla.

Con esta implementación el método que crea la actividad (onCreate) pasó de contener la implementación del modelo, de la vista y del controlador en un bloque grande de código a ser un bloque pequeño y autoexplicativo, en particular cabe destacar que el haber implementado directamente el modelo y el controlador de la pestaña que contiene la información de los últimos 30 días en la actividad (siguiendo la estructura que ya tenía esta) habría hecho que el código se volviese ilegible por su tamaño excesivo y habría generado una cantidad significativa de duplicación de código.

Luego de los cambios realizados código del método onCreate se ve como:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    context = this;
    /* it begins showing daily information */
    statistics = new DailyConnectedTimeInformation(context);
    /* set common view */
    setView();
    /* Setup tabs */
    TabHost tabHost = (TabHost) findViewById(R.id.tabHost);
    tabHost.setup();
    setTabs(tabHost);
    tabHost.setOnTabChangedListener(new OnTabChangeListener() {
    @Override
    public void onTabChanged(String tabId) {
        /* showingDay: true if we are on the DAY TAB */
        showingDay = !tabId.equals(MONTH_TAB_SPEC);
        if (showingDay) {
            statistics = new DailyConnectedTimeInformation(context);
        } else {
            statistics = new MonthlyConnectedTimeInformation(context);
        }
        loadGraphics(statistics);
        }
    });
    loadGraphics(statistics);
}
```

3.3. Implementación de las interfaces de usuario

Una vez implementado el MVC se procedió a resolver el problema del diseño gráfico en sí mismo. Anterior a este punto solo se habían implementado las clases DoughnutChartBuilder, DailyConnectedTimeInformation y DailyConnectionTypeInformation pues eran los elementos necesarios para re-implementar las vistas diarias de "Tiempo conectado" y "Tipo de señal recibida". Sin embargo el resto de las clases no fueron creadas hasta que se decidió el diseño de la pestaña mensual.

3.3.1. Proceso iterativo

El proceso de crear un diseño gráfico apropiado para la nueva pestaña se llevó a cabo mediante un proceso de desarrollo iterativo. En dicho proceso, en el área de creación de la idea visual, participaron el encargado del proyecto y la diseñadora, mientras que la practicante se encargó de la implementación de los diseños.

En cada iteración del proceso se encontraron las siguientes 3 etapas:

1. **Desarrollo del diseño:** En dicha etapa el encargado de proyecto o la diseñadora desarrollaban una idea o ilustración de como se debía ver la interfaz de la pestaña mensual. Se debe destacar que en todo momento el diseño fue pensado para implementarse tanto en "Tiempo conectado" como "Tipo de señal recibida"
2. **Implementación:** Aquí la practicante implementaba la idea del diseño utilizando tanto las clases provistas para el manejo de diseño, como con XML.
3. **Revisión del resultado:** Una vez el resultado se encontraba implementado este se mostraba al encargado del proyecto. Si el encargado no se encontraba conforme con el resultado se volvía a la primera etapa de la iteración.

Luego de la primera iteración aparecen las clases MonthlyConnectedTimeInformation y MonthlyConnectionTypeInformation las cuales sufrieron modificaciones en cada iteración en cuanto al cómo guardaban la información extraida, sin embargo la extracción misma de la información se realizó siempre de la misma manera.

3.3.2. Resultados

Finalmente se optó por un diseño de tipo ListView el cual muestra en su primera sección una barra que representa porcentualmente el uso de cada tipo de señal y que en las secciones inferiores del ListView muestra el uso en días de cada tipo de señal de forma de dejar claro a cuánto tiempo en días corresponde el uso porcentual mostrado.

Para llevar a cabo el diseño final se implementaron las clases FatBar, ListBarBuilder y FatBarBuilder, las cuales se explicarán a continuación.

1. **FatBar:** La clase FatBar se encarga de crear el "dibujo" de la barra que muestra los datos porcentuales. La clase FatBar extiende a la clase ImageView y su dibujo es creado dentro de un objeto Canvas. Para crear un tipo de gráfico FatBar este necesita una lista de valores, de colores asociados a los valores, ancho y alto de la barra. Además es posible mediante los métodos setPercentagesVisible() y setPercentagesGone() activar o desactivar la opción de mostrar a qué porcentaje númerico corresponde cada porción de la barra, en caso de hacerse visibles los porcentajes la barra requiere datos sobre el diámetro del porcentaje a mostrar, el tamaño de la letra, etc. Dichos porcentajes se muestran sólo si hay suficiente espacio para que estos sean en efecto mostrados en la porción correspondiente de la barra. Para poder determinar si el espacio en la porción de la barra es suficiente se utiliza una función del diametro que se espera que ocupe dicho porcentaje.

2. **FatBarBuilder:** FatBarBuilder corresponde a una implementación de GraphicStatisticBuilder y actúa como controlador de los gráficos tipo FatBar, es decir se encarga de pasar los datos recibidos desde el modelo a un objeto FatBar, además configura las opciones de visualización de la barra como lo es el mostrar o no los porcentajes correspondientes a cada sección de la barra.

3. **ListBarBuilder:** ListBarBuilder es un controlador, es decir implementa a GraphicStatisticBuilder, que se encarga de generar el ListView que contiene la barra en su primera sección y el uso de cada señal en días en las secciones siguientes.

Los resultados gráficos del trabajo realizado pueden verse en las figuras 4.8, 4.9, 4.10, 4.11.

Capítulo 4

Conclusiones

Los resultados del trabajo presentado en este informe fueron considerados por el encargado del trabajo como satisfactorios, siendo puestos en producción dentro de la versión 1.3b, la cual se encuentra actualmente disponible en la *play store* [4] para Android. Además gracias al trabajo realizado por la practicante en cuanto a refactoring, ahora es posible modificar el código referente a los gráficos de forma sencilla y modular, permitiendo crear nuevas extensiones sin mayores problemas.

Respecto al futuro de la aplicación hay que destacar que aún se sigue trabajando en ella realizando actualizaciones para mejorar su llegada al público en términos de visualizaciones que le provean más y mejor información al usuario, en particular se seguirá trabajando sobre las secciones modificadas por la practicante en miras de mejorar la performance de las visualizaciones mediante optimizaciones en la extracción de datos. Además las modificaciones hechas a nivel de código ayudarán a migrar fácilmente desde las pestañas implementadas como TabHost a pestañas implementadas mediante fragmentos, los cuales son ampliamente utilizados en aplicaciones para Android para el manejo de pestañas y otro tipo de vistas de forma más limpia y organizada.

Dentro del aprendizaje obtenido en el proceso de la práctica se destacan a nivel técnico el aprendizaje de Java para creación de aplicaciones para Android y sus elementos especificos como actividades, vistas, interacciones con XML y fragmentos [5]. También a nivel técnico se tiene el aprendizaje de XML para aplicaciones en Android y de git para manejo de control de versiones. Todas las tecnologías mencionadas fueron de suma importancia para llevar a cabo de forma apropiada y a tiempo el trabajo pedido a la practicante. Dentro de otros aspectos se aprendió sobre el trabajo en equipo, la importancia de mantener constantemente la comunicación para realizar un buen trabajo en particular cuando se trabaja en un equipo multidiciplinario. La practicante estuvo en constante contacto con la diseñadora y el encargado para poder obtener una visualización agradable y atractiva para el usuario de Adkintun Mobile. Finalmente queda destacar el aprendizaje sobre la importancia que tiene para el desarrollo de cualquier software que deba interactuar con un usuario el mantener una interfaz gráfica amigable, fácil de navegar y entender, y como algo que muchas veces parece tan simple como la vista puede ser determinante en cuanto a si un usuario

deseará o no usar finalmente la aplicación o software que se ha desarrollado.

Bibliografía

[1] Sitio web de NIC Chile Research Labs: http://www.niclabs.cl/

[2] Desarrollo de software para android: http://en.wikipedia.org/wiki/Android_software_development

[3] Sugar (ORM): http://satyan.github.io/sugar/

[4] Adkintun Mobile con las actualizaciones realizadas: https://play.google.com/store/apps/details?id=adk.monitor&hl=en

[5] Fragmentos: Guía de desarrollo. http://developer.android.com/guide/components/fragments.html

Figura 4.1: Vista original de "Tiempo conectado"

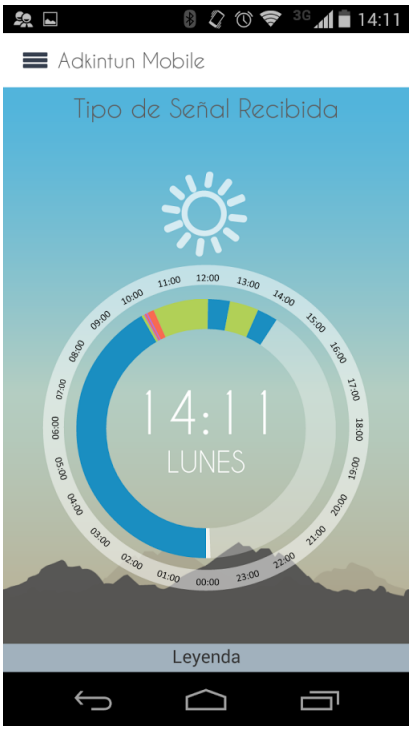

Figura 4.2: Vista original de "Tipo de señal recibida"

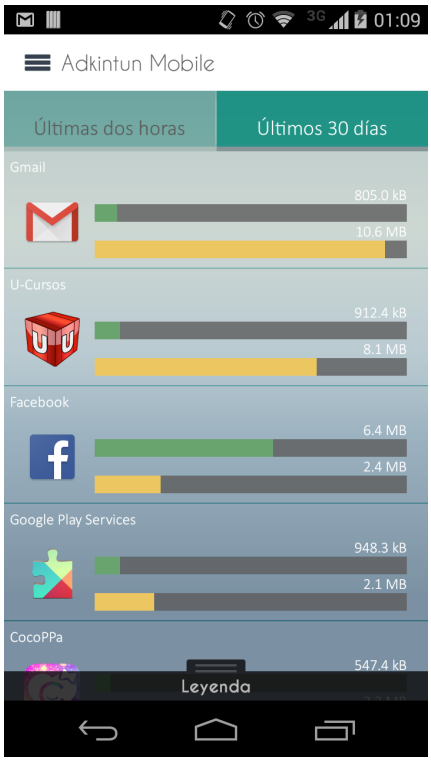

Figura 4.3: Vista de "Tráfico móvil por aplicación"

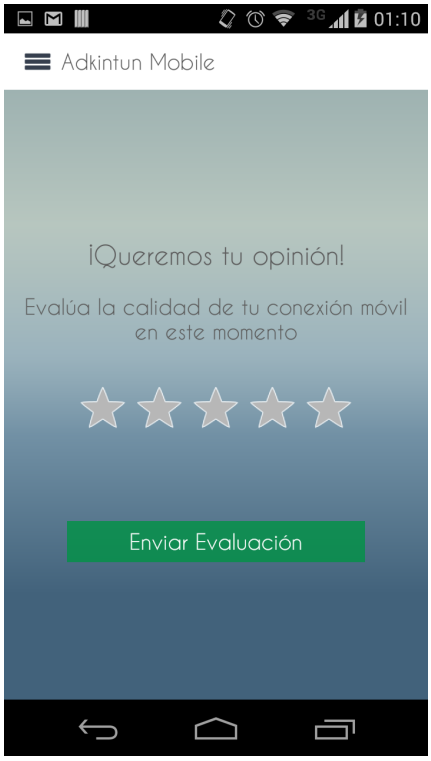

Figura 4.4: Vista de "Evalúa tu conexión"

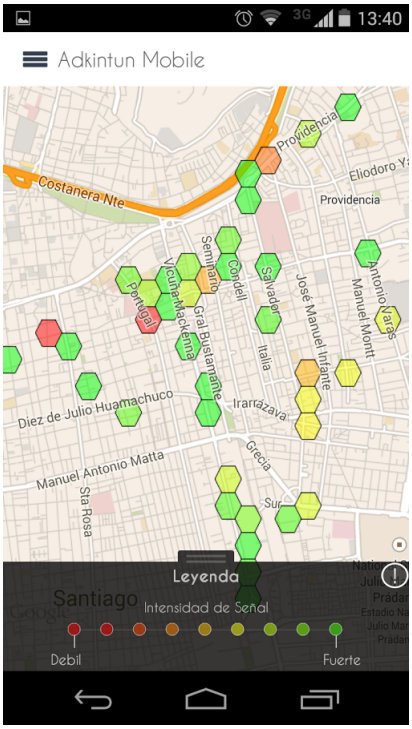

Figura 4.5: Vista de "Mapa de calidad de señal"

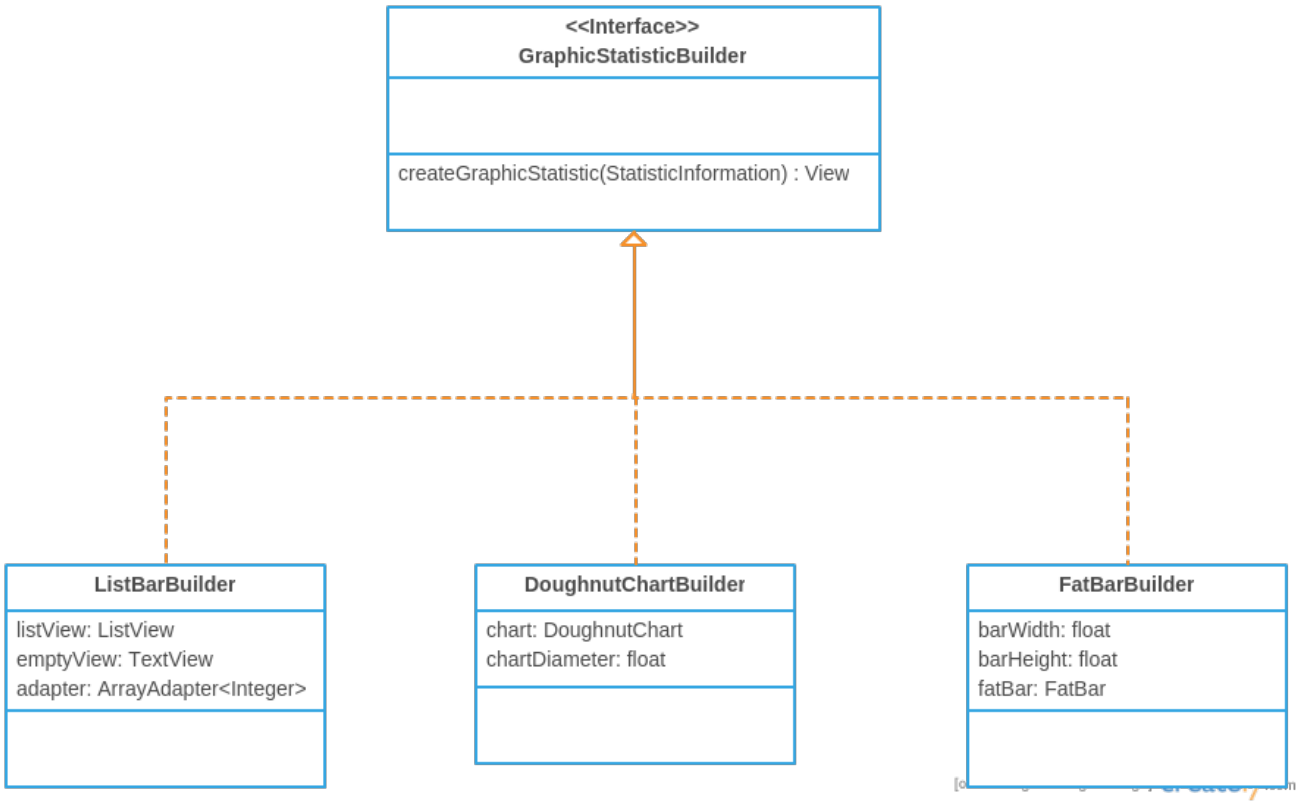

Figura 4.6: Diagrama de clases que implementan a GraphicStatisticsBuilder

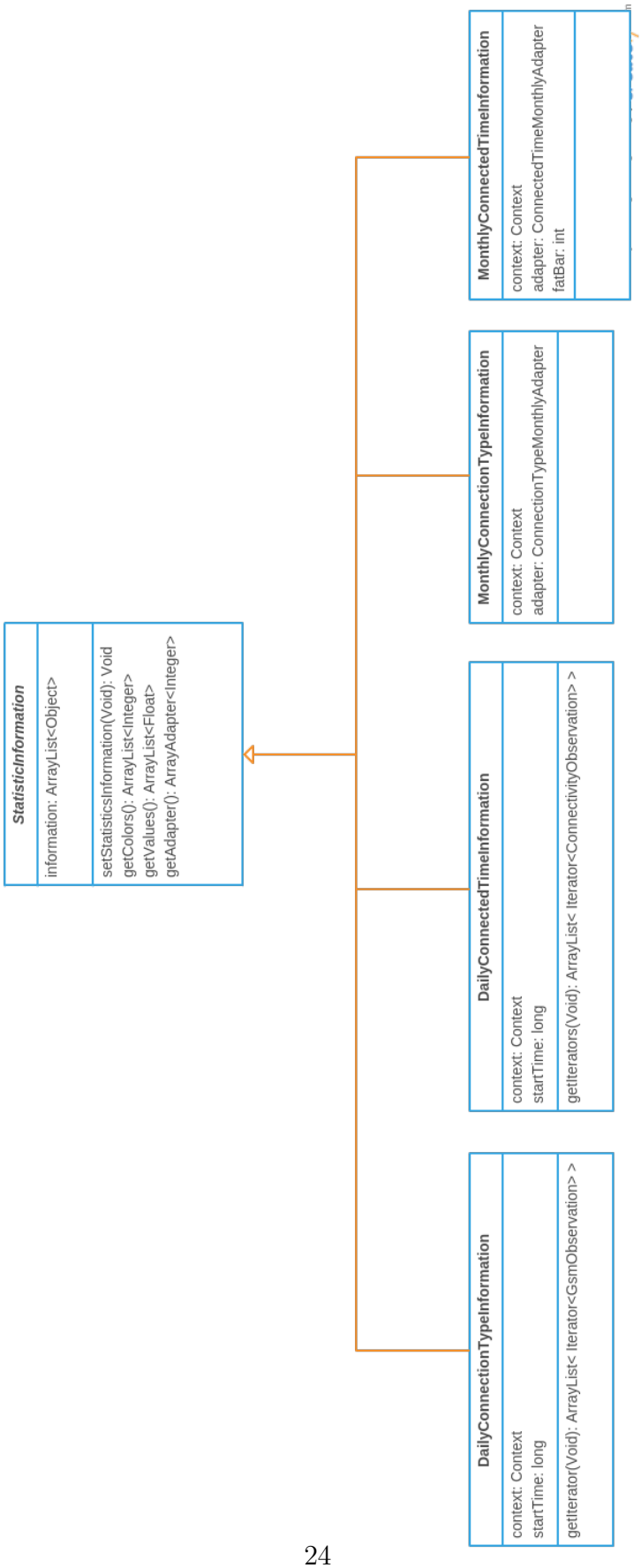

Figura 4.7: Diagrama de clases que extienden a StatisticInformation

Figura 4.8: Vista final diaria de "Tiempo conectado"

Figura 4.9: Vista final mensual de "Tiempo conectado"

Figura 4.10: Vista final diaria de "Tipo de señal recibida"

Figura 4.11: Vista final mensual de "Tipo de señal recibida"