Práctica de Patrones del Software
svg4mobile

Daniel Lahoz Titos
09025037-K
daniel.lahoz@gmail.com

Alvaro Tanarro Santamaría
01189342-N
tanarro@gmail.com

12 de mayo de 2010

Índice

1. Introducción

svg4mobile es una aplicación móvil desarrollada en Java para el sistema operativo Android 1.5 y superiores, ha sido gestada como una aplicación para el concurso MobiGame 2010 que la Universidad de Alcalá realiza todos los años en la Escuela Politécnica.

2. Enunciado

Actualmente la mayoría de dispositivos móviles, incluidos los *Smartphones* de última generación no integran en sus navegadores el estándar SVG, por lo que no pueden acceder a información guardada en esta codificación.

SVG (Scalable Vector Graphics) es una especificación para describir gráficos vectoriales en formato XML. SVG se convirtió en un estándar W3C en septiembre de 2001, por lo que ya ha sido incluido de forma nativa en la mayoría de navegadores.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<svg
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:cc="http://creativecommons.org/ns#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:svg="http://www.w3.org/2000/svg"
   xmlns="http://www.w3.org/2000/svg"
   width="735.03961"
   height="720.34869"
   id="svg2"
   version="1.1">
    <path
      style="fill:#32cd32;fill-opacity:1;fill-rule:nonzero;stroke:#000000;stroke-width:3;stroke-linecap:round;stroke-linejoin:bevel;stroke-miterlimit:4;stroke-opacity:1;stroke-dasharray:none"
      d="M 295.53125 170.5 C 295.59288 172.1545 295.625 173.83146 295.625 175.5 C 295.625 252.1161 230.09863 314.41288 148.5625 316.1875 L 148.5625 497.625 L 485.71875 497.625 L 485.71875 170.5 L 295.53125 170.5 z "
      transform="translate(1.7856969,4.2907543)"
      id="rect3762" />
    <rect
      style="fill:#ffd700;fill-opacity:1;fill-rule:nonzero;stroke:#000000;stroke-width:3;stroke-linecap:round;stroke-linejoin:bevel;stroke-miterlimit:4;stroke-opacity:1;stroke-dasharray:none"
      id="rect3837"
      width="98.571426"
      height="196.42857"
      x="54.642841"
      y="305.49658" />
    <rect
      style="fill:#ffffff;fill-opacity:1;fill-rule:nonzero;stroke:#000000;stroke-width:2.91465473;stroke-linecap:round;stroke-linejoin:bevel;stroke-miterlimit:4;stroke-opacity:1;stroke-dasharray:none"
      id="rect3788"
      width="37.94249"
      height="62.942486"
      x="56.743023"
      y="304.73962" />
```

La intención es que *svg4mobile* cubra este problema y proporcione a los usuarios de móviles con Android la capacidad de leer y navegar en mapas y planos en este estándar.

Para ello *svg4mobile* interpretara ficheros en el estándar SVG y los dibujará usando las librerías gráficas que promociona el Framework de Android. Además, *svg4mobile* proporciona una serie de opciones únicas en el caso de los mapas, que lo seguirían haciendo una aplicación útil en caso de que los navegadores móviles lo soportaran.

svg4mobile esta optimizado para planos, es capaz de orientarlos, y además puede leer información extra, como puntos de interés, descripciones o fotos.

Inicialmente se pensó en un desarrollo basado en OpenGL ES que integra la mayoría de dispositivos Android y la mayoría de móviles, sin embargo la carencia de curvas de Bézier en la versión ES nos obligo a replantear este requisito, optando finalmente por el uso de *Canvas*.

3. Requisitos

- La aplicación debe ser capaz de interpretar y pintar la figuras de tipo Text, Rect, Circle y Path de SVG.
- Debe permitir el desplazamiento por el documento.
- Debe permitir opciones zoom básicas.
- Debe permitir opciones de visualización avanzadas, como perspectiva.
- Debe ser capaz de leer opcionalmente ficheros con información extra.
- Debe ser capaz de pintar sobre el plano los puntos de interés con su fotografías correspondientes.
- Debe ser capaz de mostrarnos en una lista todas los puntos de intereses, con sus descripciones, etiquetas, fotografías, etc.
- Debe poder pintar usando los colores originales expuestos en el SVG original.
- Debe poder ejecutarse en terminales que usen Android 1.5 o superior.
- Debe proporcionar un sistema de navegación capaz de aprovechar la pantalla táctil.
- Debe ser capaz de acceder a la brújula del teléfono en caso de disponer de ella.
- Debe ser capaz de orientar el mapa usando la brújula.
- La aplicación debe ser fácilmente mantenida y estar preparada para la inclusión constante de mejoras y funcionalidades.
- La aplicación será liberada bajo una licencia libre.
- La aplicación debe ser capaz de ganar el mobigame 2010 ◡̈

4. Instrucciones ejecución

4.1. Paso 1. Preparación del ordenador de desarrollo

Requisitos:
Sistemas operativos soportados

- Windows XP (32-bit) o Vista (32- o 64-bit)
- Mac OS X 10.5.8 o superior (sólo x86)
- Linux (probado en Ubuntu y Debian)

IDE

- Eclipse 3.5 (Galileo)

JDK 6

- Android Development Tools plugin

4.2. Paso 2. Descargar el SDK Starter Package

El primer paso para desarrollar en Android es descargar el SDK Starter Package (http://developer.android.com/sdk/index.html). Asegurándose de que el paquete descargado es el correspondiente a tu entorno de desarrollo.

Después de descargar y descomprimir el SDK habrá que guardarlo en un lugar seguro en la maquina, recordando donde se encuentra ya que habrá hacer referencia al mismo al instalar el plugin ADT en Eclipse.

Opcionalmente se puede agregar al PATH del sistema las aplicaciones en la carpeta tools, localizada dentro del SDK.

4.3. Paso 3. Instalar el plugin ADT para Eclipse

En Eclipse, seleccionar Help ¿Install New Software
En Available Software, click en Add...
En Add Site poner un nombre (por ejemplo, “Android Plugin”)
En “Location” poner:

```
https://dl-ssl.google.com/android/eclipse/
```

Click en OK.

En Available Software, se dedeberá ver ahora ”Developer Tools“ añadido a la lista. Seleccionarlo y Clic en Next. Seguir las instrucciones y reiniciar Eclipse.

4.4. Paso 4. Agregar la plataforma Android y otros componentes a Eclipse

Ir al menú Window > Preferences y en la ventana que aparecerá, en el apartado de Android añadir la ruta donde se almacenó la descarga del paso 2.

En Available Packages añadir e instalar todo lo que aparece.

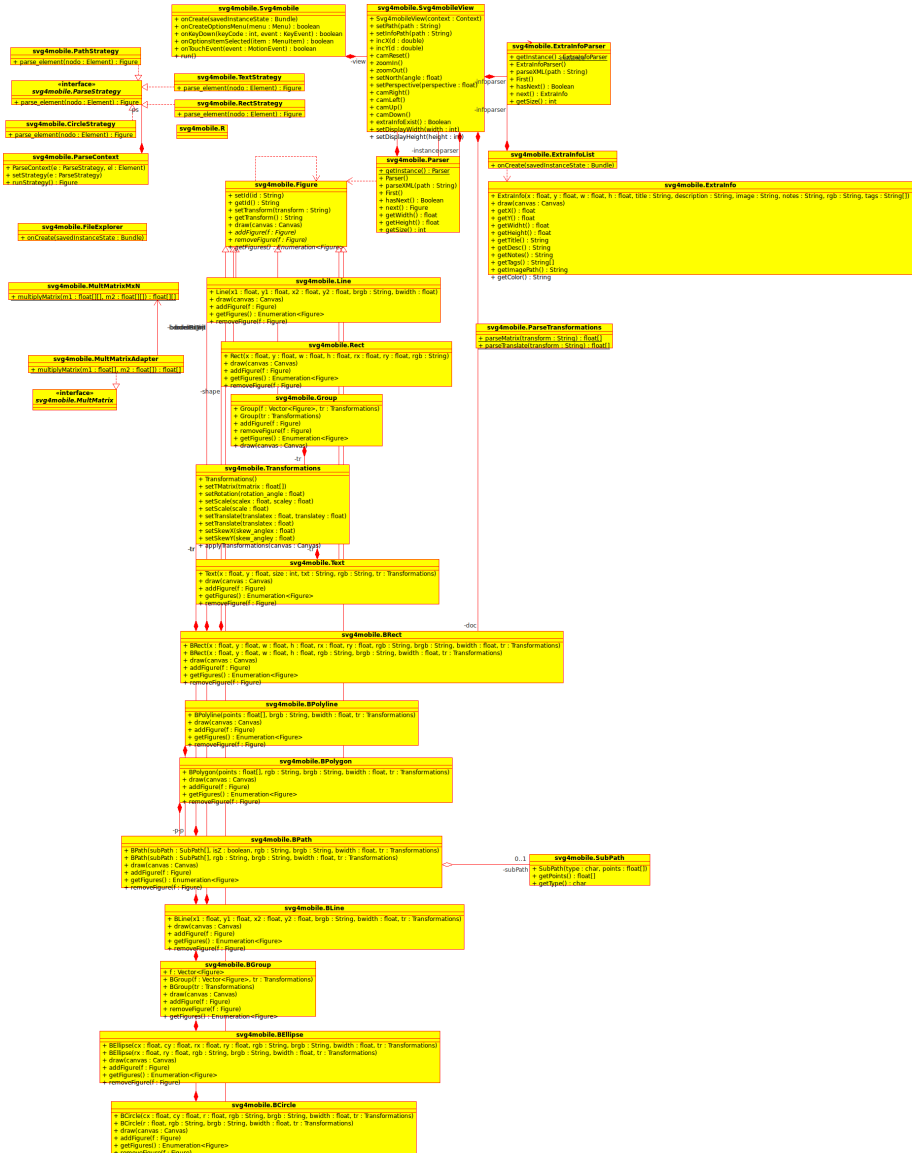
4.5. Paso 5. Importar el proyecto

File > Import > General > Existing Projects in Workspace > Next > Browse y elegir la ruta al proyecto *svg4mobile* y Finish.

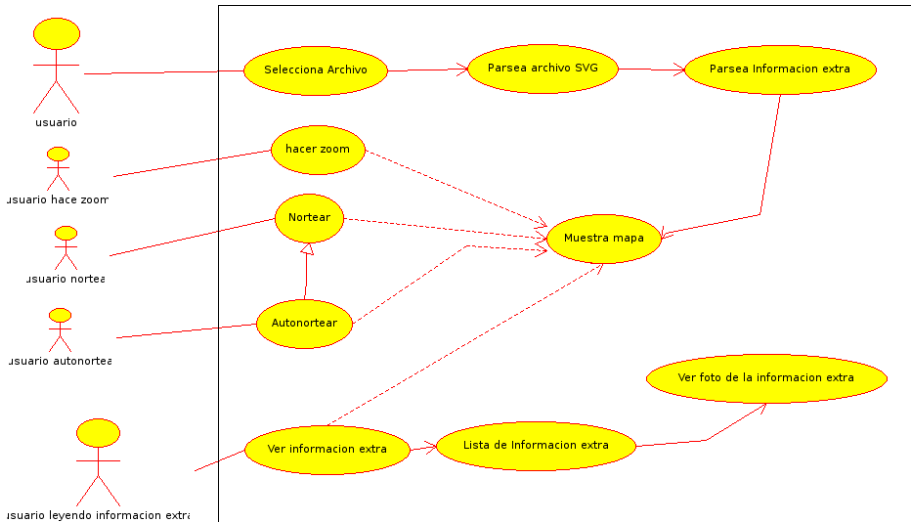
Finalmente ya preparado el entorno de desarrollo para programar aplicaciones para Android y visualizar el proyecto en concreto. Ha de tenerse en cuenta que el arranque del emulador resulta especialmente lento, sin embargo una vez que este ha sido lanzado no es necesario cargarlo múltiples veces en caso de llevar a cabo alguna modificación sobre el código.

5. Diseño completo de la aplicación (UML)

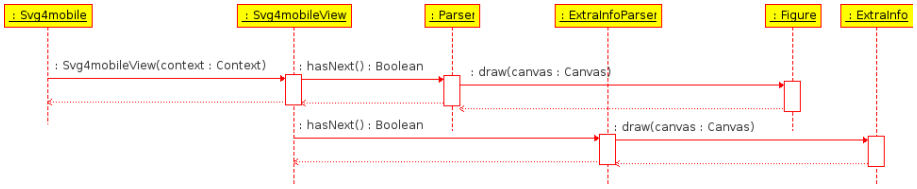
La siguiente figura muestra el diagrama de clases de toda la aplicación en general, todas las clases que posee la aplicación se encuentran aquí, con sus relaciones más importantes.



Aquí podemos ver un diagrama de casos de usos donde vemos las opciones mas importantes, como son la carga de un fichero SVG, la realización de Zoom, posicionar respecto al norte y el acceso a la información extra.



Aquí tenemos un diagrama de secuencias que muestra el proceso de lectura y parseado de un archivo SVG junto a su información extra.



6. Patrones y justificación

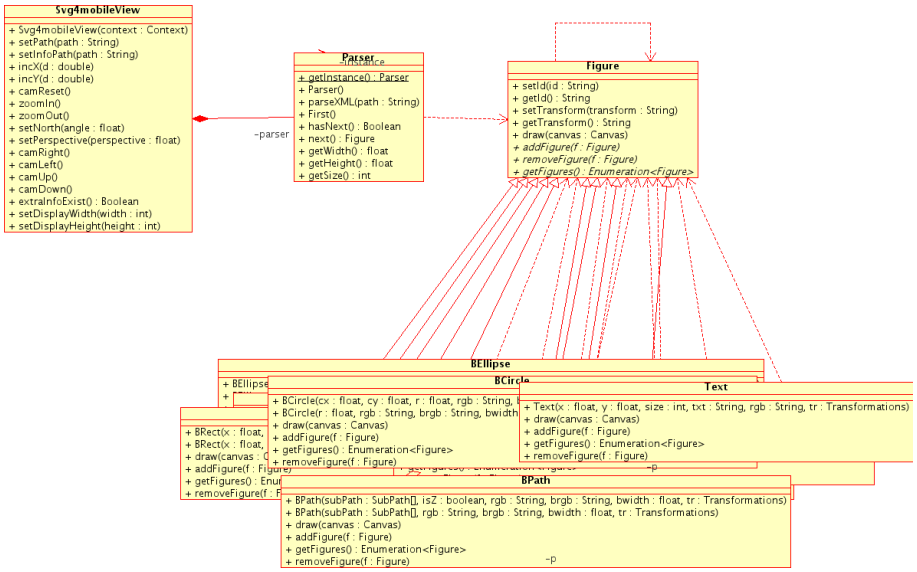
6.1. Patrón de Creación: Builder

Como tenemos que crear objetos distintos que representan a las figuras y almacenarlos y manipularlos de manera uniforme hemos empleado el patrón de creación Builder.

Las clases que intervienen en este patrón son:

- Svg4mobileView: Director
- Figure: Constructor
- BPath, BRect, BCircle, etc: Constructores concretos.

A continuación se muestra el diagrama de clases correspondiente a este patrón.



6.2. Patrón de Creación: Singleton

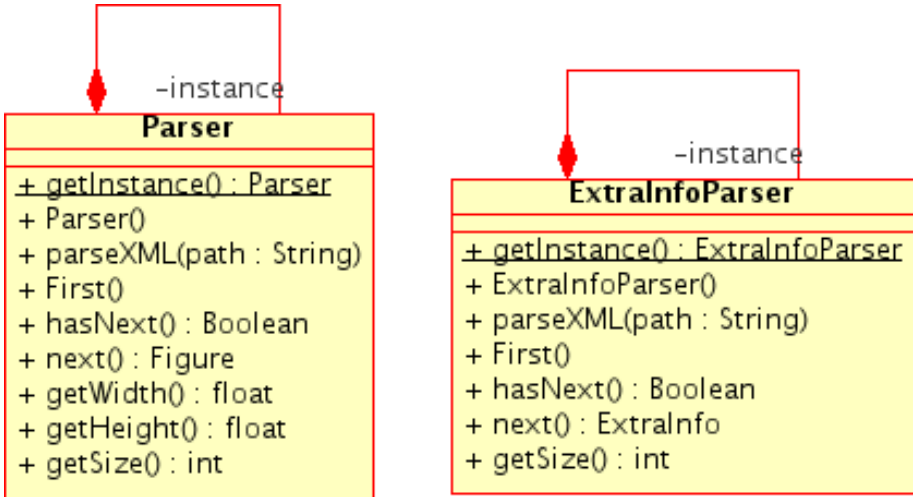
Tanto el Parser (Parser.java) como para el ExtraInfoParser (ExtraInfoParser.java) se han implementado haciendo uso del patrón Singleton, lo que nos permitido limitar su ejecución a una única instancia. Esto es algo muy necesario teniendo en cuenta que supone un proceso muy costoso y se requiere que sea ejecutado una única vez. Además al utilizar la información que genera desde varias clases, ha resultado el mecanismo mas cómodo.

Se han desarrollado los métodos getInstance() y createInstance() que devuelven la instancia y la crean respectivamente.

```
private synchronized static void createInstance() {
  if (instance == null) {
    instance = new Parser();
  }
}
```

```
private synchronized static void createInstance() {
  if (instance == null) {
    instance = new Parser();
  }
}
```

A continuación se muestra los diagramas de clases correspondientes a este patrón.



6.3. Patrón Estructural: Composite

El elemento group del estándar SVG permite agrupar elementos de cualquier tipo, lo que incluye otros grupos de manera recursiva. Para estos elementos se aplican sus atributos así como los del grupo que los contiene. En caso de que un elemento pertenezca a un grupo que a su vez pertenece a otro grupo, se deberán ir aplicando los atributos de manera recursiva con un número arbitrario de niveles de profundidad. Para solucionar este requisito, hemos optado por emplear el patrón Composite, que nos ofrece este funcionamiento gracias a su estructura en forma de árbol. En nuestro caso, un grupo sería el "contenedor" y un elemento cualquiera el "contenido".

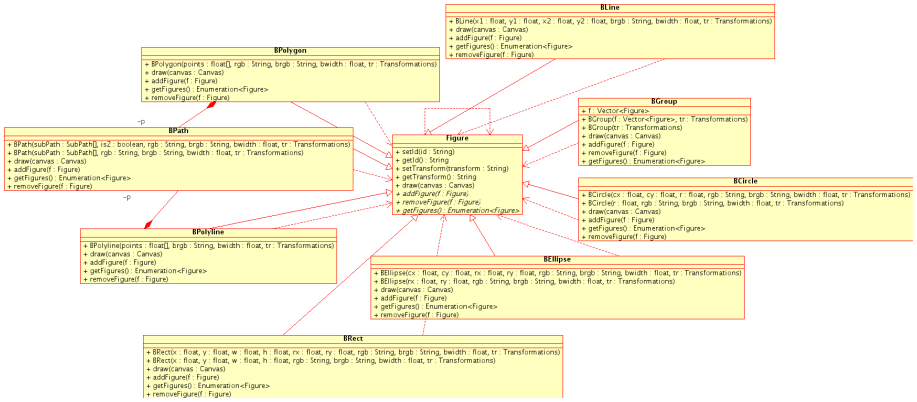
Las clases implicadas en este patrón son las siguientes:

- Figura: Componente
- BGroup: Compuesto
- BPath, BRect, BCircle, etc: Hojas
- svg4mobileView: Cliente

De esta forma, cuando el cliente ejecute el método draw() de un elemento de tipo Group, este delegará la función en el método draw() de cada uno de sus hijos (previa aplicación de los atributos del grupo al canvas):

```
canvas.save();
this.tr.applyTransformations(canvas);
for (int i=0; i<f.size(); i++)
    this.f.elementAt(i).draw(canvas);
canvas.restore();
```

A continuación se muestra el diagrama de clases correspondiente a este patrón.



6.4. Patrón Estructural: Adapter

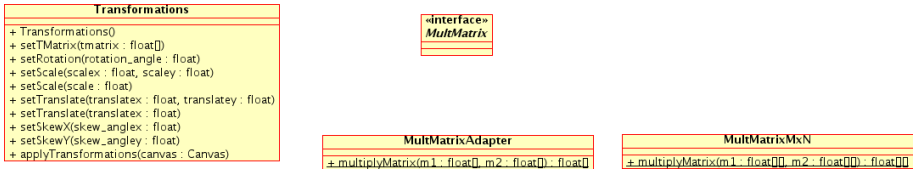
Las matrices de transformación permiten realizar cualquier transformación algebraica sobre un plano. El estandar SVG contempla varios tipos de transformación para cada elemento que se recogen en el atributo transform. Una de ellas, que de hecho engloba al resto, es la multiplicación de la matriz en uso de un elemento por una matriz de transformación. En SVG se obtienen estas matrices en forma de un array de la forma matrix(a,b,c,d,e,f. Por otro aldo, disponíamos de una clase capaz de moltiplicar matrices de la forma NxM dados dos arrays bidimensionales.

El patrón adapter nos permite adaptar las matrices tal como las obtenemos del SVG en arrays 3x3 que nuestra clase para multiplicación de matrices puede manejar.

Para la implementación de este patrón se han utilizado las siguientes clases:

- MultMatrix: Interfaz
- MultMatrixAdapter: adaptador
- MultMatrixNxM: clase a adaptar
- Transformations: Cliente

A continuación se muestra el diagrama de clases correspondiente a este patrón.

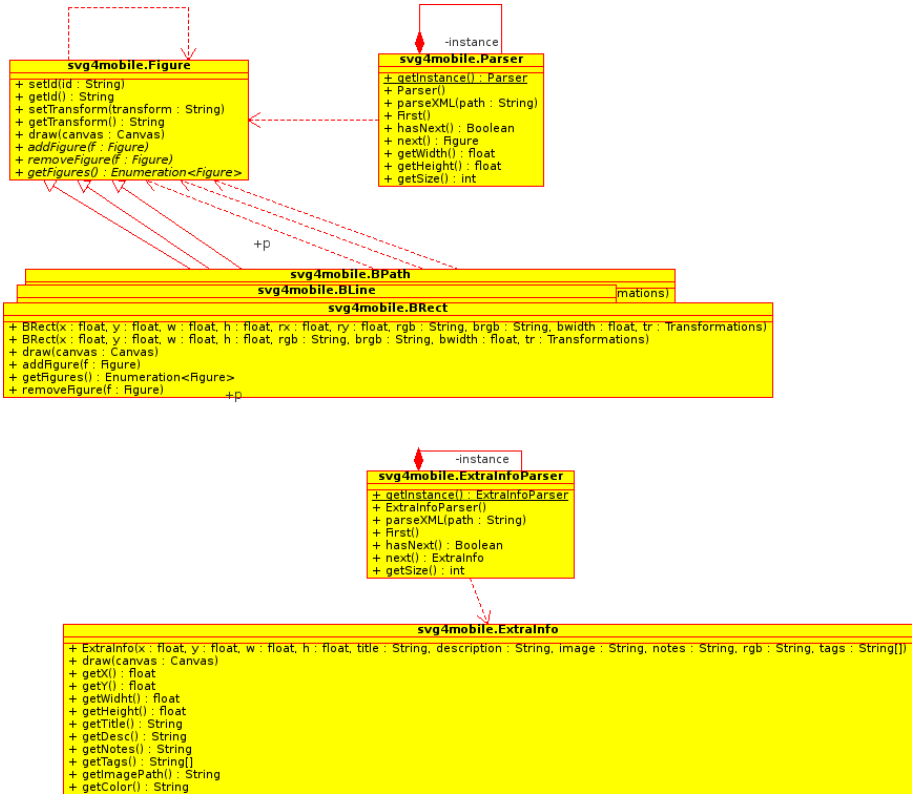


6.5. Patrón de Comportamiento: Iterator

De nuevo, tanto para el Parser (Parser.java) como para el ExtraInfoParser (ExtraInfoParser.java) se han implementado como Iteradores, ya que era conveniente disponer de un mecanismo para recorrer los elementos parseados de forma uniforme para dibujarlos. Se han implementado mediante la inclusión de los métodos First(), hasNext() y next() que permiten desplazarse por la collección.

Como el iterador recorre elementos de tipo Figure –una interfaz común para todos los elementos que se vayan parseando–, las clases que utilicen el iterador no tienen por que conocer el tipo de elementos que contiene la colección.

A continuación se muestra los diagramas de clases correspondientes a este patrón.



6.6. Patrón de Comportamiento: Strategy

Debido a las peculiaridades de cada tipo de elemento, el parser estaba creciendo de manera exagerada. Por esta razón hemos optado por utilizar un Strategy para solventarlo, así como para posibilitar que sea sencilla la implementación de nuevas estrategias concretas.

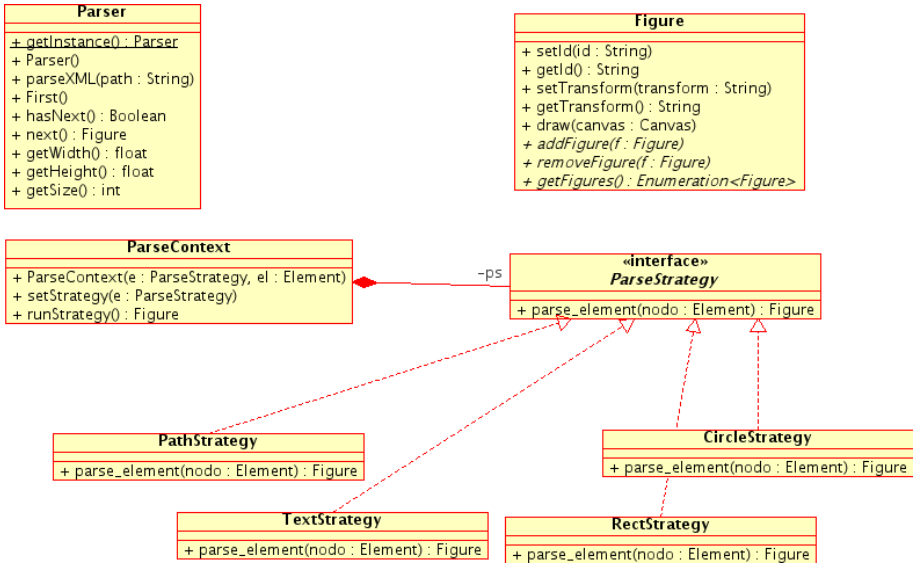
Las clases implicadas en la implementación de este patrón son:

- Parser: Cliente
- ParseContext: Contexto

- ParseStrategy: Estrategia
- PathStrategy, CircleStrategy, TextStrategy, RectStrategy, etc: Estrategias concretas

El cliente hace uso de las funciones setStrategy y runStrategy para establecer la estrategia concreta a aplicar y parsear el elemento con ella. Las estrategias concretas parsean el elemento y lo devuelven al cliente que lo añade a un Vector.

A continuación se muestra el diagrama de clases correspondiente a este patrón.



7. Referencias

1. http://code.google.com/p/svg4mobile/
2. http://www.w3.org/Graphics/SVG/
3. http://developer.android.com/index.html