libmsvg

librería C minimalista para lectura y escritura de ficheros SVG

Proyecto Fin de Máster en Software Libre

Especialidad Desarrollo de aplicaciones de software libre
Autor Mariano Álvarez Fernández
Consultor Gregorio Robles Martínez
Fecha 3/01/2011

© Mariano Álvarez Fernández
Este trabajo está bajo una licencia
Attribution-ShareAlike 3.0 Spain
de Creative Commons.
Para ver una copia de esta licencia, visite
http://creativecommons.org/licenses/by-sa/3.0/es/
o envíe una carta a Creative Commons, 171
Second Street, Suite 300, San Francisco,
California 94105, USA.

Resumen

Este documento es la memoria de un proyecto de Fin de Máster en Software Libre cursado en la Universitat Oberta de Catalunya.

El nombre del proyecto es “libmsvg” y se enmarca en la especialidad de desarrollo de aplicaciones de software libre.

“libmsvg” es una pequeña librería software, escrita en C, para leer y grabar ficheros en un subconjunto del formato SVG [1].

SVG son las iniciales de Scalable Vector Graphics, un formato de gráficos vectoriales definido por el World Wide Web Consortium [2].

En esta memoria se describe la concepción, diseño y desarrollo de la librería y los trabajos realizados para su publicación y puesta a disposición de la comunidad, como software libre [3].

Índice de contenido

Resumen....3
Capítulo 1. Introducción....6
Objetivos que se persiguen....6
Motivación y justificación....7
Área de Interés (tecnologías relacionadas)....8
Estructura de la memoria....8
Capítulo 2. Planificación....9
Alcance y tareas a realizar....9
Planificación temporal....12
Capítulo 3. Fase de desarrollo....14
Definición del subconjunto de SVG a soportar (subtarea 1.1.1)....14
Elementos a soportar en la versión 0.1....15
Elementos adicionales a soportar en la versión 1.0....18
Atributos globales a soportar en la versión 0.1....22
Atributos globales a soportar en la versión 1.0....23
Estudio de dependencias (subtarea 1.1.2)....23
Bosquejo de la estructura (subtarea 1.1.3)....24
El árbol de elementos MsvgElement....25
La estructura MsvgElement....26
Tablas de apoyo....26
Desarrollo de la librería (tarea 1.2)....27
Programas de prueba (subtarea 1.3.1)....27
Documentación inicial (subtarea 1.3.2)....28
Pruebas de portabilidad (tarea 1.4)....28
Capítulo 4. Fase de publicación....29
Planificación (tarea 2.1)....29
Web del proyecto (tarea 2.2)....29
Repositorio software (tarea 2.3)....30
Instalación de Git....31
Creación del repositorio local....32
Elección del servicio de repositorio púbico....32
Registro en GitHub y creación del repositorio....33
El primer push al repositorio público....35
Foro o lista de correo (tarea 2.4)....35
Instalación del foro....36
Aparición del SPAM....37
Publicitar proyecto (tarea 2.5)....37
Registro del proyecto en Freshmeat....37
Anuncio en la lista de correo del W3....39
Anuncio en la lista de correo de GRX....40
Documentación final (tarea 2.6)....41
Capítulo 5. Fase de mantenimiento y mejora....42
Capítulo 6. Conclusiones....43
Referencias....45

Anexo 1. Fichero readme.....47
Anexo 2. Manual del programador.....49
libmsvg 0.02 Programmer's guide.....49
Abstract.....49
Contents.....49
Starting with examples.....49
Example 1.....49
Example 2.....50
How to compile the example programs.....50
The MsvgElement tree.....51
The RAW_SVGTREE tree type.....51
The COOKED_SVGTREE tree type.....51
The FRIED_SVGTREE tree type.....52
The MsvgElement structure.....52
Reading SVG files.....52
Building a RAW MsvgElement tree by program.....53
Building a COOKED MsvgElement tree by program.....54
Manipulating a MsvgElement tree.....54
Writing SVG files.....55
Appendix A, the libmsvg SVG subset.....55

Capítulo 1. Introducción

El objetivo de este proyecto es el desarrollo de una pequeña librería software para leer y grabar ficheros en un subconjunto del formato SVG y su puesta a disposición, como software libre, a la comunidad.

El nombre del proyecto y de la librería es “libmsvg”. Si diseccionamos este nombre, “lib” significa librería software (library); “m” quiere significar mínima (minimalista) en varios sentidos, por una parte que será una librería pequeña, por otra que no tendrá ninguna o muy pocas dependencias y finalmente que sólo tratará un subconjunto del formato que se quiere soportar; “svg” hace referencia al formato que se pretende leer y grabar, Scalable Vector Graphics [1], el formato de gráficos vectoriales especificado por el World Wide Web Consortium [2].

Objetivos que se persiguen

La librería tendrá las siguientes características principales:

- **Será software libre bajo licencia LGPL v2** [4]. La elección de la licencia está motivada por el deseo de que los desarrollos añadidos por otros sigan siendo software libre, pero se quiere también maximizar su compatibilidad con cualquier otro tipo de licencia. Creo que es el mejor modelo para que una librería de software de base aumente su aceptación, pues puede ser combinada con cualquier clase de programa. Escojo además la versión 2 por una cuestión práctica, pues existe mucho código que es LGPLv2-only, pero no añado ninguna restricción para que puedan usarse las versiones 2.1 y 3 de la LGPL en trabajos derivados.
- **Estará escrita en ANSI-C**. Se pretende que sea completamente portable, para ello nada mejor que utilizar ANSI-C como lenguaje de programación, por estar presente de forma casi universal en cualquier plataforma y es sencillo crear recubrimientos para otros lenguajes. No obstante es imposible soportar la librería en cualquier plataforma, por lo que habrá un grupo pequeño de plataformas directamente soportadas.
- **No tendrá ninguna o muy pocas dependencias**. Se quiere que sea de fácil uso y sencilla de instalar y distribuir, para que pueda ser utilizada en programas sin muchos requerimientos. Para ello es necesario, al ser una librería de base, que tenga el mínimo de dependencias posible.

En la publicación, encaminada a poner la librería a disposición de la comunidad de software libre, se persigue:

- Construir un sitio WEB que será el eje central del proyecto, permitiendo la descarga de versiones y enlazando el resto de recursos.
- Utilizar una herramienta pública de control de versiones.
- Montar un foro o una lista de correo.
- Generar una documentación que invite a utilizar la librería.
- Publicitar el proyecto.

Motivación y justificación

Mi primer contacto con un proyecto de software libre fue DJGPP [5], la versión del compilador GCC para el sistema operativo DOS. DJGPP tenía una lista de correo bastante activa en la que solía participar. DJGPP utilizaba un sencillo sistema de paquetes, a base de ficheros zip. Empaqueté para DJGPP tres proyectos muy populares: "netpbm" [6] un conjunto de utilidades para transformar entre sí distintos formatos gráficos de mapa de bits (raster); "libjpeg" la librería del IJG [7] para leer y grabar ficheros en formato JPEG; y "libpng" la librería canónica [8] para leer y grabar ficheros en formato PNG.

GRX [9] es una librería de gráficos 2D desarrollada originalmente para DJGPP y convertida después en multiplataforma. Hice varias contribuciones para GRX como las funciones para leer/grabar imágenes JPEG y PNG (utilizando los paquetes que había construido para DJGPP).

Hacia mediados del año 2000 GRX quedó huérfano y a principios de 2001 di el paso y comencé a mantener la librería hasta finales del 2003, cuando por diversos motivos, lo dejé. Posteriormente construí una versión reducida de GRX con sólo el API nativo en C para 4 únicas plataformas: DOS/DJGPP, Win32/Mingw, Linux/framebuffer y Linux/X11 y la publiqué en mi web a finales de 2006 con el nombre MGRX [10].

Una de las cosas que siempre quise añadir a GRX/MGRX consistía en la posibilidad de leer y guardar ficheros gráficos vectoriales. Puestos a utilizar un formato abierto parece lógico pensar en SVG.

Añadir a GRX la posibilidad de leer y guardar ficheros en formato JPEG y PNG fue muy fácil, porque existían sendas librerías libres de referencia para ambos formatos (libjpeg y libpng), escritas en ANSI-C y casi sin dependencias (libpng depende sólo de libz, libjpeg no tiene dependencias).

Busqué algo similar, en C, para el formato SVG, pero no lo encontré. Las

librerías que leen ficheros SVG suelen estar muy imbricadas en cada proyecto concreto, por ejemplo la librería librsvg [11] de GNOME tiene múltiples dependencias y está actualmente enfocada a Cairo [12] (la librería 2D de GNOME).

Este proyecto trata de llenar éste vacío, pero en lugar de crear una librería específica para GRX/MGRX tratará de crear una librería genérica, en C, que pueda ser utilizada por múltiples proyectos. No obstante parte del proyecto consistirá en los desarrollos necesarios para usar la nueva librería en GRX/MGRX.

Área de Interés (tecnologías relacionadas)

Respecto al desarrollo:

- Programación en C
- Ficheros XML
- Gráficos vectoriales, formato SVG

Respecto a la publicación:

- Desarrollo de sitio WEB
- Sistemas de control de versiones
- Foros y Listas de correo
- Publicitar proyectos de software libre

Estructura de la memoria

El siguiente capítulo de esta memoria describe la planificación efectuada para la realización del proyecto.

Los siguientes tres capítulos están dedicados a las tres fases del proyecto: desarrollo, publicación y mantenimiento.

Se añade un último capítulo con las conclusiones del trabajo.

Se completa la memoria con una sección de referencias y dos anexos. El primer anexo contiene el fichero “readme” que describe sucintamente el proyecto e incluye las instrucciones de instalación. El segundo anexo contiene la guía del programador.

Capítulo 2. Planificación

Alcance y tareas a realizar

En esta sección se definen las diferentes tareas a realizar y se fija su alcance. Existen dos fases principales que se dividirán en tareas y subtareas, la primera fase, desarrollo, se completará durante el primer semestre (P1) y la segunda fase, publicación, durante el segundo semestre (P2). Una tercera fase, mantenimiento y mejora, se iniciará una vez acabado el desarrollo durante las vacaciones y proseguirá posteriormente en paralelo con la tarea de publicación.

Fase 1. Desarrollo, esta fase se descompone en tres tareas: planificación, desarrollo de la librería, desarrollos anexos y pruebas de portabilidad.

Tarea 1.1 Planificación, se divide a su vez en tres subtareas: definición del subconjunto de SVG a soportar, estudio de dependencias y bosquejo de la estructura.

Subtarea 1.1.1 Definición del subconjunto SVG a soportar. La especificación de SVG es muy amplia, se trata de soportar sólo un subconjunto, dicho subconjunto se elegirá dentro de las características (features) estáticas de la especificación SVG Tiny 1.2 [13].

Subtarea 1.1.2 Estudio de dependencias. Se desea que libmsvg tenga las mínimas dependencias posibles para que sea portátil. No obstante al ser SVG un lenguaje XML puede acelerarse el desarrollo utilizando alguna librería existente, bien enlazando con ella, importando su código o utilizando las ideas generales. Deben estar escritas en C y con licencia compatible con la LGL. Dentro de esta tarea se estudiarán las siguientes:

- libexpat [14], una librería muy utilizada para interpretar XML, escrita en C y con licencia MIT, compatible con LGPL. Su funcionamiento está orientado a corrientes de datos (streams).
- libxml2 [15], librería escrita en C y también con licencia MIT. Es la librería XML del proyecto GNOME. Es bastante grande y tiene varias dependencias.
- tinyxml [16], sería la librería perfecta para el proyecto, salvo que está escrita en C++. Es pequeña y su planteamiento,

respecto a los ficheros XML, es muy similar a lo que pretende libmsvg con los ficheros SVG, lee y escribe ficheros XML a una clase fácilmente tratable por programa. La licencia es la de la librería "zlib", compatible con LGPL.

- libsvg [17], fue un intento de extraer de librsvg la parte no dependiente de Cairo ni otras librerias (salvo libxml2), no pasó de beta y no está en mantenimiento, aunque aún se puede encontrar. Está pensada para "renderizar", se le pasa un conjunto de funciones para ejecutar dicho renderizado. No está en línea con este proyecto pero puede ser una fuente de ideas. La licencia es LGPL.

Subtarea 1.1.3 Bosquejo de la estructura. La idea central de libmsvg gira en torno a la definición de una estructura (struct de C) que represente el gráfico y sea fácilmente manipulable por los programas que usen la librería. Esta actividad consiste en un bosquejo general de dicha estructura.

Tarea 1.2 Desarrollo de la librería. Una vez completadas las tareas de planificación se comenzará el desarrollo de la librería propiamente. Incluye la definición final de la estructura y las rutinas de soporte para transformar un fichero SVG en dicha estructura y viceversa.

Tarea 1.3 Desarrollos anexos, contiene dos subtareas:

Subtarea 1.3.1 Programas de prueba, uno para leer un fichero SVG y mostrar una salida textual de la estructura; otro para crear un gráfico vectorial y grabarlo como fichero SVG.

Subtarea 1.3.3 Documentación inicial. Se construirá una documentación mínima en inglés de la librería que acompañará a la primera versión

Tarea 1.4 Pruebas de portabilidad. Todo el desarrollo principal se realizará con el sistema operativo linux. Dentro de esta tarea se probará que la librería compila correctamente en dos sistemas adicionales: bajo DOS usando el compilador DJGPP y bajo Windows utilizando el compilador Mingw.

Fase 2. Publicación. En concordancia con el documento “Desarrollo de proyectos de software libre” [12] la publicación y puesta a disposición de la comunidad de un proyecto de software libre, requiere de una gestión específica que cubra aspectos como: web del proyecto, repositorio software, publicación

de versiones, soporte a la comunicación de la comunidad, seguimiento de errores, etcétera. Estos aspectos se cubren con las siguientes tareas:

Tarea 2.1 Planificación. Define el alcance del resto de tareas. Aunque existen herramientas genéricas que cubren todas las necesidades de un proyecto de software libre, como SourceForge [19], voy a optar por utilizar herramientas específicas para cada aspecto de la gestión por dos motivos: primero, tener un mayor control que me permita usar herramientas sencillas y cambiarlas en el futuro: segundo, porque pretendo utilizar Git [20] como sistema de control de versiones.

Tarea 2.2 Web del proyecto. La web del proyecto es la pieza central de su publicación. Va a cubrir cuatro objetivos básicos:

- Ser el punto de entrada para conocer el objetivo, motivación y estado del proyecto.
- Recopilar la información que apunte al resto de recursos del proyectos, como foro o lista de correos, sistema de control de versiones, etcétera.
- Permitir la descarga de versiones de la librería.
- Disponer de la documentación del proyecto y hacerla visible en línea.

Para albergar la web voy a utilizar un subdominio de mi dominio particular: http://libmsvg.fgrim.com
Usaré el idioma inglés para la web, con objeto de ampliar su público objetivo: otros desarrolladores interesados en usar y participar en la librería.

Tarea 2.3 Repositorio software. Uno de los ejes que dirige esta fase es utilizar Git como sistema de control de versiones. Mi opinión es que es el sistema ideal para el trabajo colaborativo en un proyecto de software libre, pues cada posible desarrollador mantiene su propio repositorio y sólo es necesario hacer push y pull de los cambios que hagan los demás si los consideras adecuados.
El primer paso de esta tarea será crear un repositorio local en mi propio ordenador. El segundo paso es elegir alguno de los servidores que ofrecen repositorios Git en línea gratuitos para proyectos de SL. En principio voy a decidir entre estos dos:

- http://repo.or.cz/, uno de los pioneros en ofrecer el servicio.
- http://github.com/, más moderno y muy utilizado, tiene una interfaz web más elaborada que el anterior y ofrece servicios adicionales al repositorio.

Tarea 2.4 Foro o lista de correo. Para poder vehicular la participación de otros desarrolladores en la librería, es necesario disponer de un medio

de comunicación, que puede ser un foro o una lista de correo cuyas conversaciones puedan archivarse y seguirse en línea. Para completar esta tarea se valorarán alternativas como registrar un foro o lista de correo en algún servidor público o instalar uno en mi dominio.

Tarea 2.5 Publicitar proyecto. El objeto de esta tarea es dar a conocer la librería. El primer paso será registrar el proyecto en Freshmeat [21] que es el canal de anuncios de software libre por antonomasia. Posteriormente se irán añadiendo anuncios de nuevas versiones de la librería. El siguiente paso es identificar listas de correo y foros relacionados con programación de librerías gráficas y el formato SVG y enviar mensajes anunciando el proyecto.

Tarea 2.6 Documentación final. Uno de los puntos débiles de la mayoría de proyectos de software libre es la documentación, que suele ser escasa, desactualizada y poco didáctica. Está tarea irá encaminada a obtener un buen tutorial que guíe a los nuevos programadores.

Fase 3. Mantenimiento y mejora. Esta fase se inicia tras el fin del desarrollo, paralelamente a la fase 2, y se mantendrá activa durante toda la vida del proyecto, idealmente más allá de la finalización del master, e incluirá posibles mejoras de otros. Aunque el foco de la segunda parte del proyecto es la publicación, es fundamental seguir mejorando la librería e incluir posibles aportaciones de terceros. Esto generará nuevas noticias para actualizar la web y añadir mensajes sobre los canales de información del proyecto.
Como tarea incluida en esta fase se construirá un programa de prueba para GRX/MGRX con objeto de renderizar ficheros SVG. Será un programa muy básico para desarrollar posteriormente.

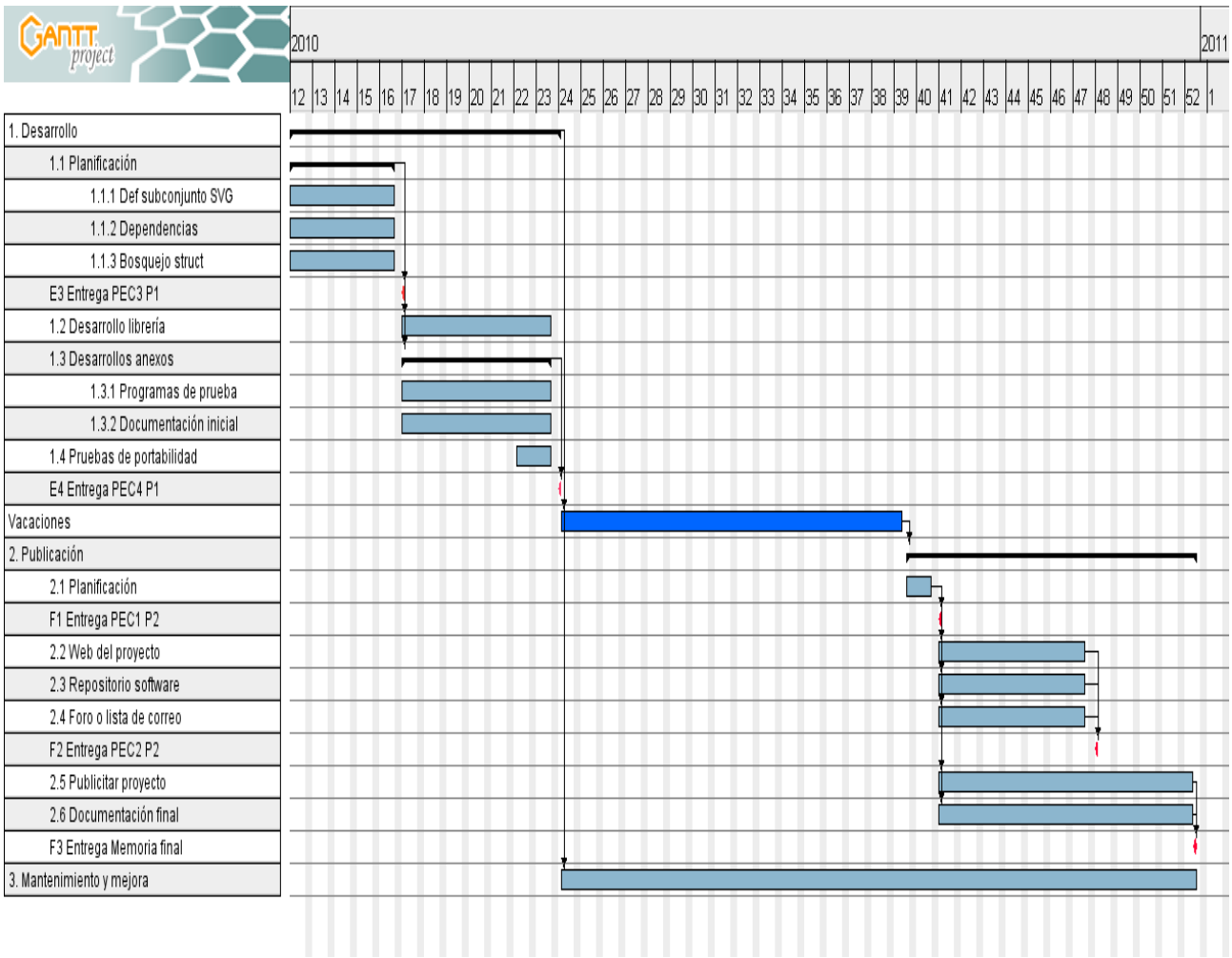
Planificación temporal

En el gráfico que se acompaña, desarrollado con la herramienta de gestión de proyectos GanttProject [22], se muestra la planificación temporal de las distintas fases y tareas del proyecto.

En la fase de desarrollo se arranca con la tarea de planificación y sus tres subtareas de forma paralela, tras acabar ésta se lanzan las tareas de desarrollo de la librería y desarrollos anexos, por último la tarea de pruebas de portabilidad se programa al final de la fase. Se incluyen dos hitos: E3 y E4, las entregas de la PEC3 y de la PEC4 de la primera parte del proyecto de fin de Máster. Requieren la construcción de un documento de estado del proyecto y enviarlo al tutor de la asignatura.

En la fase de publicación, tras la tarea inicial de planificación, se lanzan el resto de tareas de forma paralela. Existen tres hitos enmarcados en la segunda parte del proyecto de fin de Máster. F1 corresponde a la entrega de la PEC1 y F2 a la entrega de la PEC2. Ambos implican la construcción de un informe intermedio de progreso. El hito F3 exige el desarrollo de esta memoria final del proyecto.

La fase de mantenimiento y mejora arranca tras el final de la fase de desarrollo y será una labor continua, en paralelo con la fase de publicación.



Capítulo 3. Fase de desarrollo

La fase de desarrollo se divide en cuatro tareas: planificación (1.1), dividida a su vez en tres subtareas; desarrollo de la librería (1.2); desarrollos anexos (1.3), compuesta de dos subtareas; y pruebas de compatibilidad (1.4).

Todas estas tareas y subtareas se describen en las siguientes secciones. Esta fase se completó durante el primer semestre de este PFM.

Definición del subconjunto de SVG a soportar (subtarea 1.1.1)

SVG es un lenguaje XML para describir gráficos en dos dimensiones. Permite tres tipos de objetos gráficos: formas gráficas vectoriales, multimedia y texto. Los objetos pueden agruparse y asignarles estilo y transformaciones. Los documentos SVG pueden ser estáticos o interactivos y dinámicos. Las animaciones pueden definirse de forma declarativa o vía "scripts".

Esta implementación contempla sólo documentos SVG estáticos que incluyan sólo un subconjunto de dos de los tipos de objetos: formas gráficas vectoriales y texto. Durante la lectura de un documento SVG se ignorarán todas las características no soportadas que pudiera contener.

El subconjunto a soportar estará basado en la especificación SVG Tiny 1.2 [13].

Se va a definir aquí explícitamente el subconjunto de elementos y atributos SVG soportados. Se distinguirá además entre el subconjunto de elementos y atributos que se soportarán en la versión 0.1 de la librería y el subconjunto adicional que se pretende soportar cuando la librería alcance la versión 1.0.

Debe notarse que la versión actual de la librería es la 0.02 y aún no soporta todos los atributos previstos para la 0.1, aunque sí los elementos.

El subconjunto para una futura versión 1.0 de la librería es lo bastante completo para soportar la mayoría de dibujos SVG estáticos. Se describe aquí porque la intención es seguir desarrollando la librería más allá de la finalización de este trabajo de fin de Máster.

En cada elemento se relacionan los atributos soportados bajo 3 clases:

- particulares, son atributos necesarios para definir el elemento.
- opcionales, son atributos que pueden aparecer o no con el elemento.
- globales, son grupos de atributos que se describirán de forma genérica para todos los elementos que lo soporten después de la lista de elementos.

Para cada elemento se listan los tipos de elementos hijo que pueden contener.

Aunque no es propiamente un atributo, se usará en la lista el atributo "contenido" para almacenar el contenido de aquellos elementos que puedan tenerlo.

Elementos a soportar en la versión 0.1

Elemento <svg>. Es el elemento raíz que contiene el dibujo vectorial

Atributos particulares:
- viewBox="min-x min-y width height". Identifica el tamaño del gráfico en coordenadas de usuario, los valores pueden estar separados por comas y/o espacios.
- width="width". Define el ancho en las unidades que se especifiquen (pixels, cm, ...).
- height="height". Define el alto en las unidades que se especifiquen (pixels, cm, ...).

Atributos opcionales:
- version="valor"
- baseProfile="valor"
- preserveAspectRatio="valor"
- viewport-fill="color". Define el color de fondo (ver atributo genérico fill).
- viewport-fill-opacity="n". Define la opacidad, de 0 (transparente) a 1 (opaco).

Posibles elementos hijos:
<g>, <rect>, <circle>, <ellipse>, <line>, <polyline>, <polygon>

Posibles elementos hijos adicionales en v1.0:
<title>, <desc>, <defs>, <use>, <path>

Elemento <g>. Es un elemento contenedor, agrupa otros elementos gráficos.

Atributos particulares:
- id="valor"

Atributos globales:
- transformaciones
- estilos de dibujo
- estilos de texto

Posibles elementos hijos:
<g>, <rect>, <circle>, <ellipse>, <line>, <polyline>, <polygon>

Posibles elementos hijos adicionales en v1.0:
<title>, <desc>, <use>, <path>

Elemento <rect>. Define un rectángulo.

Atributos particulares:
- x="n". Origen-x
- y="n". Origen-y
- width="n". Ancho >= 0
- height="n". Alto >= 0

Atributos opcionales:
- id="valor"
- rx="n". x-radio para redondear esquinas
- ry="n". y-radio para redondear esquinas

Atributos globales:
- transformaciones
- estilos de dibujo

Elemento <circle>. Define un círculo.

Atributos particulares:
- cx="n". Centro-x
- cy="n". Ccentro-y
- r="n". Radio

Atributos opcionales:
- id="valor"

Atributos globales:
- transformaciones
- estilos de dibujo

Elemento <ellipse>. Define una elipse.

Atributos particulares:
- cx="n". Centro-x
- cy="n". Ccentro-y
- rx="n". Radio-x
- ry="n". Radio-y

Atributos opcionales:
- id="valor"

Atributos globales:
- transformaciones
- estilos de dibujo

Elemento <line>. Define una línea.

Atributos particulares:
- x1="n". Posición-x esquina superior izquierda
- y1="n". Posición-y esquina superior izquierda
- x2="n". Posición-x esquina inferior derecha
- y2="n". Posición-y esquina inferior derecha

Atributos opcionales:
- id="valor"

Atributos globales:
- transformaciones
- estilos de dibujo

Elemento <polyline>. Define una polilínea.

Atributos particulares:
- points="datos". Parejas x,y de la polilínea

Atributos opcionales:
- id="valor"

Atributos globales:
- transformaciones
- estilos de dibujo

Elemento <polygon>. Define un polígono.

Atributos particulares:
- points="datos". Parejas x,y del polígono

Atributos opcionales:
- id="valor"

Atributos globales:
- transformaciones
- estilos de dibujo

Elementos adicionales a soportar en la versión 1.0

Elemento <defs>. Es un elemento contendor, que contiene elementos referenciables, pero que no se muestran hasta ser referenciados, es equivalente a una librería de elementos de dibujo.

Posibles elementos hijos:
<g>, <title>, <desc>, <path>, <rect>, <circle>, <ellipse>, <line>, <polyline>, <polygon>, <solidColor>, <linearGradient>, <radialGradient>

Elemento <title>. Provee un título informativo a un elemento.

Atributos particulares:
- contenido="valor"

Elemento <desc>. Provee una descripción informativa a un elemento.

Atributos particulares:
- contenido="valor"

Elemento <use>. Usa un elemento definido previamente y referenciado mediante el atributo "id".

Atributos particulares:
- x="coordenada". Posición x que se considera como origen.
- y="coordenada". Posición y que se considera como origen.
- xlink:href="IRI". Id del elemento referenciado.

Atributos globales:

- transformaciones
- estilos de dibujo
- estilos de texto

Elemento <path>. Define un contorno o forma (outline o shape).

Atributos particulares:
- d ="path-data". Una serie de comandos define el contorno o forma:
  M x y ---> mueve a x,y (absoluto)
  m x y ---> mueve a x,y (relativo)
  L x y ---> línea a x,y (absoluto)
  l x y ---> línea a x,y (relativo)
  Z ---> cierra la línea al último M ó m
  z ---> cierra la línea al último M ó m
  H x ---> linea horizontal a x (absoluto)
  h x ---> linea horizontal a x (relativo)
  V y ---> linea vertical a y (absoluto)
  v y ---> linea vertical a y (relativo)
  C, c, S, s ---> curvas bezier cúbicas
  Q, q, T, t ---> curvas bezier cuadráticas
- pathLength="n". Longitud del path (es una ayuda)

Atributos opcionales:
- id="valor"

Atributos globales:
- transformaciones
- estilos de dibujo

Elemento <text>. Define una línea de texto.

Atributos particulares:
- contenido="valor"
- x="coordenada". Posición x que se considera como origen.
- y="coordenada". Posición y que se considera como origen.
- rotate="angulo". Rotación a aplicar a cada carácter individual.

NOTA. Estos tres atributos pueden tener un valor único o una lista de valores que se irán aplicando a cada carácter del texto.

Atributos opcionales:
- id="valor"

Atributos globales:
- transformaciones
- estilos de dibujo
- estilos de texto

Posibles elementos hijos:
<tspan>

Elemento <tspan>. Este elemento puede aparecer dentro de un elemento <text> para asignar estilos específicos a un trozo de texto.

Atributos particulares:
- contenido="valor"

Atributos globales:
- transformaciones
- estilos de dibujo
- estilos de texto

Elemento <textArea>. Define una línea de texto que se inscribe en un rectángulo.

Atributos particulares:
- contenido="valor"
- x="coordenada". Posición x que se considera como origen.
- y="coordenada". Posición y que se considera como origen.
- width="n". Ancho >= 0
- height="n". Alto >= 0

Atributos opcionales:
- id="valor"
- line-increment="valor"
- text-align ="valor"

Atributos globales:
- transformaciones
- estilos de dibujo
- estilos de texto

Posibles elementos hijos:
<tbreak>

Elemento <tbreak>. Es un elemento sin contenido ni atributos que indica un salto de

línea dentro del texto de un elemento textArea.

Elemento <solidColor>. Define un color sólido que puede referenciarse posteriormente por atributos que esperan un color.

Atributos particulares:
- id="valor"
- solid-color="color". (Ver atributo genérico fill).
- solid-opacity="n". Opacidad, de 0 a 1.

Elemento <linearGradient>. Define un gradiente lineal que puede referenciarse posteriormente por atributos que esperan un color.

Atributos particulares:
- id="valor"
- gradientUnits="valor". Puede ser "userSpaceOnUse" para usar el sistema de coordenadas en uso o "objectBoundingBox" para usar un sistema de coordenadas basado en el rectángulo que envuelve el objeto al que aplica el gradiente, de (0,0) a (1,1). Por defecto es "objectBoundingBox".
- x1="coordenada"
- y1="coordenada"
- x2="coordenada"
- y2="coordenada". Estos cuatro atributos definen el vector que dirigirá el gradiente.

Posibles elementos hijos:
<stop>

Elemento <radialGradient>. Define un gradiente radial que puede referenciarse posteriormente por atributos que esperan un color.

Atributos particulares:
- id="valor"
- gradientUnits="valor". (Ver elemento linearGradient)
- cx="coordenada"
- cy="coordenada"
- r="radio". Estos tres atributos definen el centro y radio que dirigirá el gradiente.

Posibles elementos hijos:
<stop>

Elemento <stop>. Define un color para un punto de parada de un gradiente lineal o radial.

Atributos particulares:

- offset="n". Define el punto de parada dentro del vector o radio que dirige le gradiente.
- stop-color="color". Color del punto de parada.
- stop-opacity="n". Opacidad de 0 a 1 del punto de parada.

Atributos globales a soportar en la versión 0.1

Atributo transform (transformaciones). Este atributo define una transformación de coordenadas, toda transformación se puede representar como una matriz de transformación [23]. En la versión 0.1 se soportará una sola transformación por atributo de las siguientes:

- transform="translate(x,y)". Traslación.
- transform="rotate(angulo)". Rotación.
- transform="scale(factor)". Escalar.

Atributo fill (estilo de dibujo). Define el color del relleno de una forma. Puede ser un color en formato hexadecimal de 3 cifras "#rgb", hexadecimal de 6 cifras "#rrggbb", funcional entero "rgb(rrr,ggg,bbb), funcional real "rgb(r%,g%b%)", un nombre de color (ej: "black"), un color de sistema (ej: "ActiveBorder") o una referencia a un elemento solidColor, linearGradient ó radialGradient. En v0.1 se soportarán sólo las formas hexadecimal y nombre de color.

Atributo fill-opacity (estilo de dibujo). Define la opacidad del relleno de una forma, de 0 (transparente) a 1 (opaco).

Atributo stroke (estilo de dibujo). Define el color del controno de una forma (ver atributo fill).

Atributo stroke-width (estilo de dibujo). Define el ancho del contorno de una forma.

Atributo stroke-opacity (estilo de dibujo). Define la opacidad del contorno de una forma, de 0 (transparente) a 1 (opaco).

Atributos globales a soportar en la versión 1.0

Atributo transform (transformaciones). En la versión 1.0 se soportará una lista de transformaciones en un único atributo transform y las siguientes transformacioens adicionales:

- transform="skewX(angulo)". Inclinar respecto al eje X.
- transform="skewY(angulo)". Inclinar respecto al eje Y.
- transform="matrix(a,b,c,d,e,f)". Define directamente la matriz de transformación.

Atributo direction (estilo de texto). Define la dirección del texto, puede valer "ltr" ó "rtl".

Atributo text-anchor (estilo de texto). Define como se alinea el texto respecto al punto origen.

Atributo font-family (estilo de texto). Define la familia de la fuente.

Atributo font-style (estilo de texto). Define el estilo, puede ser "normal", "italic" u "oblique".

Atributo font-weight (estilo de texto). Define el peso de la fuente, "normal", "bold", etc.

Atributo font-size (estilo de texto). Define el tamaño de la fuente.

Estudio de dependencias (subtarea 1.1.2)

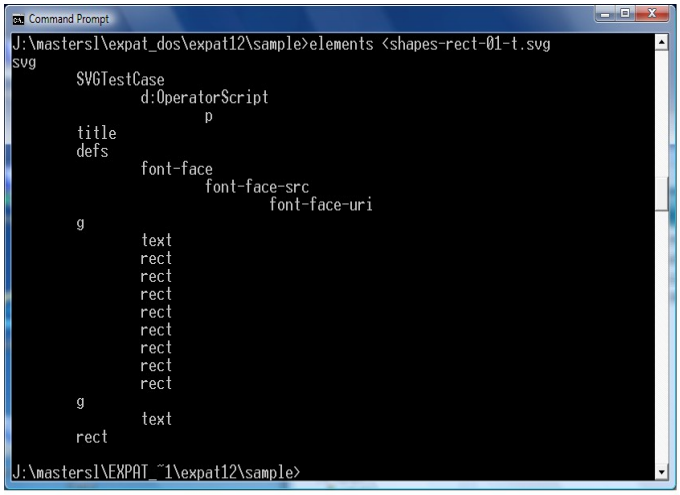
Se desea que libmsvg tenga las mínimas dependencias posibles para que sea portátil. No obstante al ser SVG un lenguaje XML puede acelerarse el desarrollo utilizando alguna librería existente, bien enlazando con ella, importando su código o utilizando las ideas generales. Deben estar escritas en C y con licencia compatible con la LGPL.

En el capítulo de planificación mencioné 3 librerías candidatas para interpretar XML: libexpat [14], libxml2 [15] y tinyxml [16].

Respecto a tinyxml, aun siendo a priori la que más se acercaba a lo que necesitaba, está escrita en C++, por lo que requeriría bastante trabajo realizar una versión en C.

Para comprobar la viabilidad de usar una de las dos restantes, he intentado la instalación en uno de los entornos más limitados que pretendo soportar, el compilador djgpp [5] para DOS. Una instalación básica de djgpp carece del shell y del conjunto de herramientas típicas de un sistema linux, por lo que no es factible usar la maquinaria de las autotools.

La compilación de libxml2 requiere usar un script de configuración que no funciona directamente en una instalación básica de djgpp. Al ser la librería muy grande, construir un Makefile específico es un trabajo notable.



Igualmente libexpat basa su compilación en un script de configuración, aunque es una librería más pequeña. Existe además una versión antigua de libexpat, la 1.27, la última que liberó el autor original de la librería, que basa su compilación en un sencillo Makefile. Ha sido muy fácil modificarlo para obtener una versión de la librería para djgpp. En la imagen anexa se ve la salida de un pequeño programa de prueba que muestra, indentados, los elementos de un fichero xml, en este caso un fichero svg seleccionado de la TestSuite [24] que mantiene el W3C.

La decisión final es utilizar la versión 1.2 de libexpat incluyéndola dentro del proyecto libmsvg, con lo que tendremos cero dependencias.

Libexpat no incluye mucha documentación, pero existe un pequeño tutorial [25] de la versión 1.1, con algunos programas de ejemplo, que muestra claramente como usar la librería.

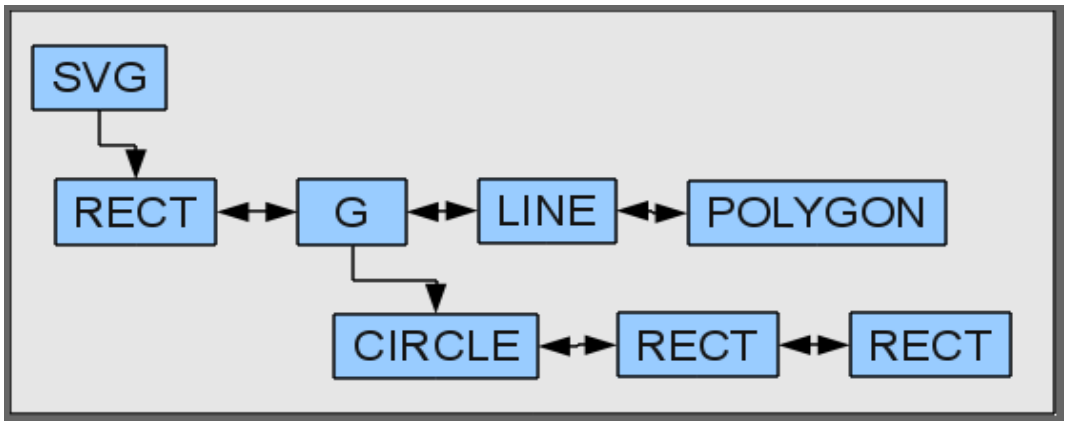
Bosquejo de la estructura (subtarea 1.1.3)

La idea central de libmsvg gira en torno a la definición de una estructura (struct de C) que represente el gráfico y sea fácilmente manipulable por los programas que usen la librería. Esta actividad consiste en un bosquejo general de dicha estructura.

El árbol de elementos MsvgElement

En libmsvg un gráfico SVG se representa como un árbol de elementos MsvgElement. Cada elemento tiene un id que define el tipo de elemento (eid), un puntero al elemento padre, punteros a sus hermanos precedente y siguiente y un puntero a su primer hijo. El elemento raíz debe ser del tipo EID_SVG. En las primeras versiones de la librería sólo los elementos EID_SVG y EID_G pueden tener elementos hijos. Los otros elementos soportados son EID_RECT, EID_CIRCLE, EID_ELLIPSE, EID_LINE, EID_POLYLINE y EID_POLYGON.

El siguiente gráfico muestra un ejemplo de un árbol del elementos MsvgElement.



Cada elemento puede tener atributos que pueden ser de dos tipos: genéricos o específicos. El tipo de atributos en un árbol concreto es determinado por una variable en el nodo raíz. Si esta variable es RAW_SVGTREE, todos los atributos son genéricos. Los atributos genéricos son simples pares clave-valor. Si la variable es COOKED_SVGTREE, todos los atributos son específicos. Los atributos específicos se almacenan en variables con el tipo adecuado y son específicas a cada eid.

Cuando un gráfico SVG se carga en un árbol de elementos MsvgElement desde un fichero se marca con el tipo RAW_SVGTREE. Sólo los elementos soportados se insertan en el árbol, pero todos los atributos de esos elementos se almacenan como atributos genéricos. Este estado puede ser suficiente para algunos programas. Los elementos y sus atributos pueden ser añadidos, borrados y reordenados y finalmente escritos en un nuevo fichero.

Usando una función de la librería un árbol del tipo RAW_SVGTREE puede convertirse al tipo COOKED_SVGTREE. Sólo los atributos soportados son convertidos a atributos específicos. Este estado es mucho más fácil de manipular por un programa que pretenda renderizar el gráfico.

La estructura MsvgElement

Esta es la versión final de la estructura que se define en el fichero msvg.h:

```
typedef struct _MsvgElement *MsvgElementPtr;

typedef struct _MsvgElement {
  enum EID eid;
  MsvgElementPtr father; /* pointer to father element */
  MsvgElementPtr psibling; /* pointer to previous sibling element */
  MsvgElementPtr nsibling; /* pointer to next sibling element */
  MsvgElementPtr fson; /* pointer to first son element */
  MsvgAttributePtr fattr; /* pointer to first generic attribute */
  union { /* specific attributes */
    MsvgSvgAttributes *psvgattr;
    MsvgGAttributes *pgattr;
    MsvgRectAttributes *prectattr;
    MsvgCircleAttributes *pcircleattr;
    MsvgEllipseAttributes *pellipseattr;
    MsvgLineAttributes *plineattr;
    MsvgPolylineAttributes *ppolylineattr;
    MsvgPolygonAttributes *ppolygonattr;
  };
} MsvgElement;
```

Los atributos genéricos se almacenan en una sencilla lista encadenada de variablesMsvgAttribute:

```
typedef struct _MsvgAttribute *MsvgAttributePtr;

typedef struct _MsvgAttribute {
  char *key; /* key attribute */
  char *value; /* value attribute */
  MsvgAttributePtr nattr; /* pointer to next attribute */
} MsvgAttribute;
```

Los atributos específicos son diferentes para cada tipo de elemento y se almacenan como una unión.

Tablas de apoyo

Se crearán varias tablas para dirigir la interpretación de la lectura del documento svg:

- La tabla de elementos, que asigna un id a cada elemento e indica si está soportado o no.
- La tabla de herencia, indica si un elemento puede ser hijo de otro elemento.
- La tabla de atributos específicos, indica si un atributo está soportada para un elemento dado.

Desarrollo de la librería (tarea 1.2)

Aunque de pequeño tamaño, la versión 0.01 de libmsvg, construida durante el primer semestre de este PFM, es elegante y ha requerido muchas horas de desarrollo. La librería usa exclusivamente ANSI-C por lo que es, a priori, totalmente transportable a cualquier entorno.

El código fuente se estructura en un directorio principal y cuatro subdirectorios:

- directorio principal, contiene el fichero "readme", la licencia y el Makefile que dirige la construcción de la librería.
- subdirectorio expat, contiene una versión de la librería expat con un Makefile específico para libmsvg. El incluir el código de expat dentro de la librería permite que libmsvg no tenga ninguna dependencia.
- subdirectorio src, aquí están los ficheros fuentes y el fichero de cabecera "msvg.h" que define el API de la librería.
- subdirectorio test, contiene los programas de prueba.
- subdirectorio doc, destinado a la documentación.

El desarrollo se ha realizado en un ordenador con sistema operativo GNU-Linux, utilizando el editor kate, el compilador gcc, y las herramientas gmake y ar.

El corazón de la librería es el árbol de elementos "MsvgElement" que define completamente un objeto SVG. La versión 0.01 soporta todos los elementos planeados para la futura versión 0.1, pero trata los atributos sólo como atributos genéricos (parejas key-value), por lo que sólo puede manejar árboles SVG del tipo definido como RAW.

La librería, en su versión 0.01, compila correctamente sin errores y es utilizable. Puede descargarse comprimida en versión tar.gz ó zip desde la página web del proyecto.

Programas de prueba (subtarea 1.3.1)

Se han creado dos programas de prueba en el subdirectorio test:

- t1.c, lee un fichero SVG que se le pasa como parámetro y lo convierte en un árbol de elementos "MsvgElement". Muestra dicha estructura por pantalla y la vuelve a escribir al fichero "msvgt1.svg". Dado que libmsvg ignora todo lo que no entiende del fichero de entrada, el fichero de salida incluye sólo lo

interpretado. Para facilitar el uso de este programa se incluye en el subdirectorio test un fichero SVG de prueba.

- t2.c, construye por programa un árbol de elementos "MsvgElement" y lo escribe en el fichero "msvgt2.svg".

Documentación inicial (subtarea 1.3.2)

Se completó la siguiente documentación en inglés:

- fichero readme, describe brevemente la librería y muestra como compilarla e instalarla en los tres entronos soportados.

- fichero user_manual_v0.01.html, una pequeña introducción al uso de la librería, germen de lo que será la guía del programador en la versión 0.02.

- fichero supported_elements_v0.1.html, una tabla en html de los elementos y atributos que se desean soportar en la versión 0.1 de libmsvg. Esta tabla fue incluida posteriormente en la guía del programador.

Pruebas de portabilidad (tarea 1.4)

El propósito de esta tarea fue el comprobar que la librería compilaba correctamente no sólo bajo Linux sino también en dos sistemas adicionales: bajo DOS usando el compilador DJGPP y bajo Windows utilizando Mingw.

Tras modificar los Makefiles la librería compila y funciona correctamente en los tres entornos. Sólo es necesario modificar un parámetro en el fichero "makedefs" para especificar el entorno antes de lanzar el comando make.

Capítulo 4. Fase de publicación

La fase de publicación se divide en seis tareas: planificación (2.1); web del proyecto (2.2); repositorio software (2.3); foro o lista de correo (2.4); publicitar proyecto (2.5); y documentación final (2.6).

Se describe cada tarea en las siguientes secciones. Esta fase se completó durante el segundo semestre de este PFM.

Planificación (tarea 2.1)

En concordancia con el documento "Desarrollo de proyectos de software libre" [18] la publicación y puesta a disposición de la comunidad de un proyecto de software libre, requiere de una gestión específica que cubra aspectos como: web del proyecto, repositorio software, publicación de versiones, soporte a la comunicación de la comunidad, seguimiento de errores, etcétera.

Existen herramientas genéricas que cubren todas las necesidades de un proyecto de SL como SourceForge [19]. No obstante he optado por utilizar herramientas específicas para cada aspecto de la gestión por dos motivos: primero, tener un mayor control que me permita usar herramientas sencillas y cambiarlas en el futuro: segundo, porque pretendo utilizar Git [20] como sistema de control de versiones.

Durante esta tarea se planificó el resto de tareas de la fase de publicación.

Web del proyecto (tarea 2.2)

La web del proyecto es la pieza central de su publicación. Cubre cuatro objetivos básicos:

- Ser el punto de entrada para conocer el objetivo, motivación y estado del proyecto.
- Recopilar la información que apunta al resto de recursos del proyecto, como el foro, el repositorio software, etcétera.
- Permitir la descarga de versiones de la librería.
- Disponer de la documentación del proyecto y hacerla visible en línea.

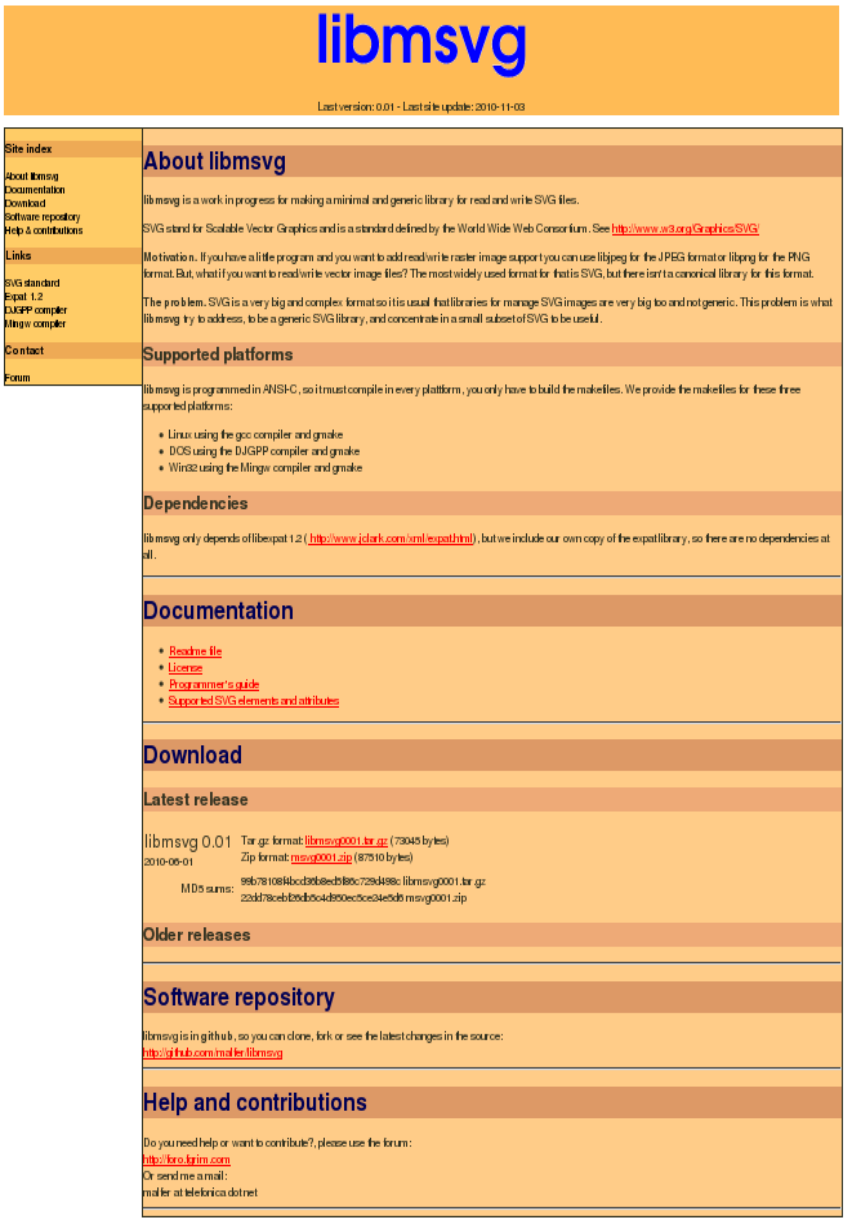
Para albergar la web he utilizado un subdominio de mi dominio particular:

http://libmsvg.fgrim.com

La web está construida con un único fichero "index.html" dividido en tres secciones ( ): cabecera, menú y contenido.

Una hoja de estilo "libmsvg.css" se ocupa de dar formato a cada sección y disponerla en la pantalla.

La sección de contenido se divide en capítulos: about, documentation, download, software repository, help and contributions.



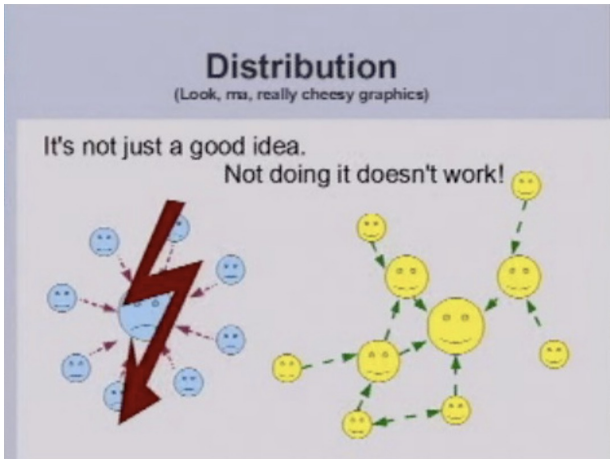
He utilizado el idioma inglés para la web con objeto de ampliar su público objetivo: otros desarrolladores interesados en usar y participar en la librería.

Repositorio software (tarea 2.3)

Uno de los ejes que dirige esta fase es utilizar Git como sistema de control de versiones. Este sistema fue desarrollado por Linus Torvalds [26] en el año 2005.

Hasta el año 2002, Linus no usaba ningún sistema de control de versiones para el desarrollo del kernel de Linux, trabajando básicamente con parches diff recibidos por correo. En el 2002 Linus comenzó a usar Bitkeeper, sistema comercial, pero con una licencia especial para los desarrolladores del kernel. Tras varios años de continuas quejas y problemas por parte de numerosos desarrolladores, en 2005 Linus dejó de utilizarlo y creó en un par de semanas Git, el sistema de control de versiones

utilizado actualmente para el desarrollo del kernel y cada vez en más proyectos de software libre.



La primera característica de Git es que no tiene un repositorio central, cada desarrollador tiene su propia versión completa y funcional del repositorio. Cuando tiene listo un conjunto de cambios pide al responsable del subsistema que "tire" (pull) de los cambios. Esto no sólo funciona hacia arriba, sino también hacia abajo y hacia los lados, de forma que varios desarrolladores pueden trabajar colaborativamente en un subsistema y sólo subir los cambios hacia arriba cuando lo consideran listo. Igualmente un desarrollador "tira" de los cambios del repositorio de nivel superior cuando lo necesita. En la imagen de la izquierda, extraida de una presentación de Linus del 2007 [27], se representa visualmente esta idea.

Otra idea central de Git es la cuestión de la "confianza". Cada persona confía en un pequeño grupo de personas, que a su vez tendrá su pequeño grupo de confianza, de esta forma los cambios fluyen entre los repositorios individuales de Git hacia el repositorio canónico sin que sea necesario que todo el mundo sincronice directamente con él y asegurando un mayor grado de seguridad en el origen y la revisión de los cambios.

Otras características de Git:

- Es muy rápido, es una colección de pequeñas utilidades escritas en C.
- Genera parches para enviar por correo, básico si el desarrollador no dispone de "confianza".
- Registra adecuadamente los cambios de nombre y ubicación de ficheros.
- Soporta versionado de ficheros binarios.

Mi opinión es que es el sistema ideal para el trabajo colaborativo en un proyecto de software libre, pues cada posible desarrollador mantiene su propio repositorio y sólo es necesario hacer push y pull de los cambios que hagan los demás si los consideras adecuados.

Instalación de Git

En las distribuciones linux modernas es fácil instalar Git usando el gestor de

paquetes propio de la distribución, En mi caso con Opensuse 11.3 basta arrancar Yast e instalar los paquetes “git” y “gitk”, este último es un programa gráfico muy conveniente que muestra la estructura del repositorio con los diferentes commit y los cambios realizados.

Una vez instalado conviene iniciar las variables globales de “user.name” y “user.email” utilizando el comando “git config –global”.

Creación del repositorio local

Previamente a la creación del repositorio local es conveniente construir, en el directorio raíz del software, un fichero “.gitignore” con lo que no queremos que aparezca en el repositorio, como los archivos “.o”, ejecutables, etc. Este es el mío:

```
# Ignore objects and archives
*.[oa]
# Ignore expat programs
expat/gennmtab/gennmtab
expat/sample/elements
# Ignore test programs
test/t1
test/t2
# Ignore graphics files from test programs
test/msvgt1.svg
test/msvgt2.svg
```

En el siguiente log comentado se muestra la creación del repositorio local:

```
mariano@zeravla5:~> cd libmsvg  <== Ir al directorio donde está el software
mariano@zeravla5:~/libmsvg> git init  <== Crea e inicia el repositorio
Initialized empty Git repository in /home/mariano/libmsvg/.git/
mariano@zeravla5:~/libmsvg> git add .  <== Añadir todo
mariano@zeravla5:~/libmsvg> git commit -m 'first commit'  <== Primer commit
[master (root-commit) 058cee1] first commit
 42 files changed, 12713 insertions(+), 0 deletions(-)
 create mode 100644 .gitignore
 create mode 100644 Makefile
 create mode 100644 copying.txt
 create mode 100644 doc/supported_elements_v0.1.html
 create mode 100644 doc/user_manual_v0.1.html
 create mode 100644 expat/Makefile
...
 create mode 100644 src/wtsvgf.c
 create mode 100644 test/Makefile
 create mode 100755 test/shapes-rect-01-t.svg
 create mode 100755 test/t1.c
 create mode 100644 test/t2.c
mariano@zeravla5:~/libmsvg> git tag -a v0.01  <== Marca el commit como versión 0.01
```

Elección del servicio de repositorio púbico

Entre los servicios que ofrecen repositorios Git públicos gratuitos para proyectos de software libre he elegido GitHub [28] por varias razones:

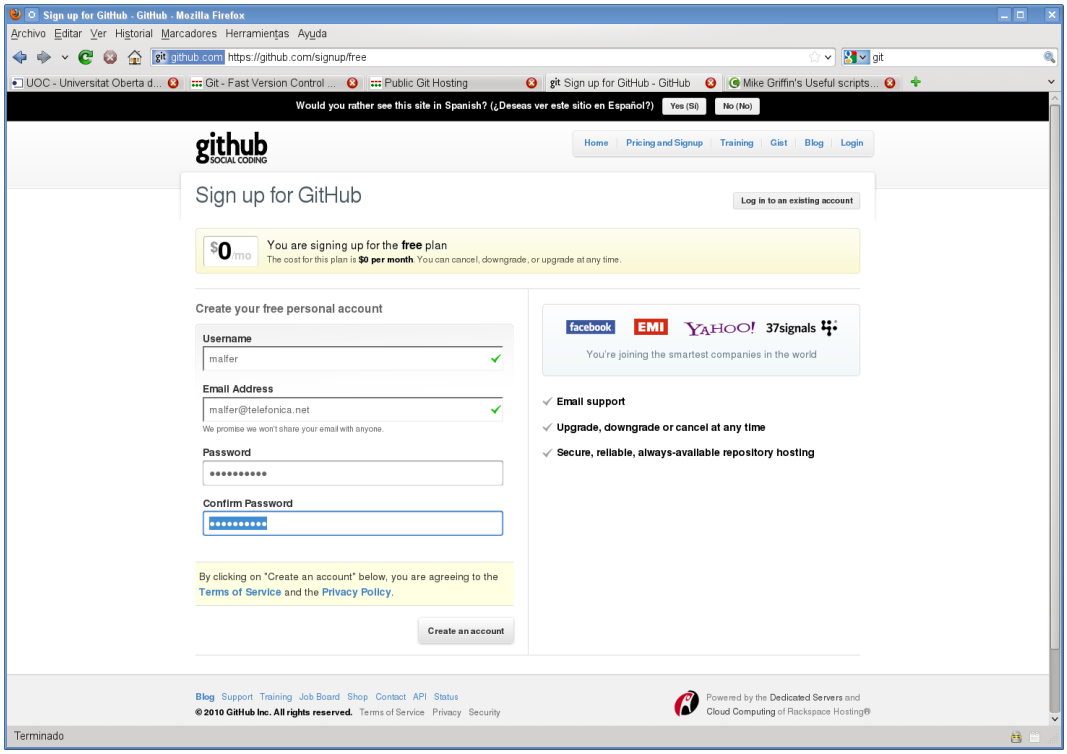
- Tiene una interfaz web muy cuidada y es muy popular entre proyectos de

software libre.

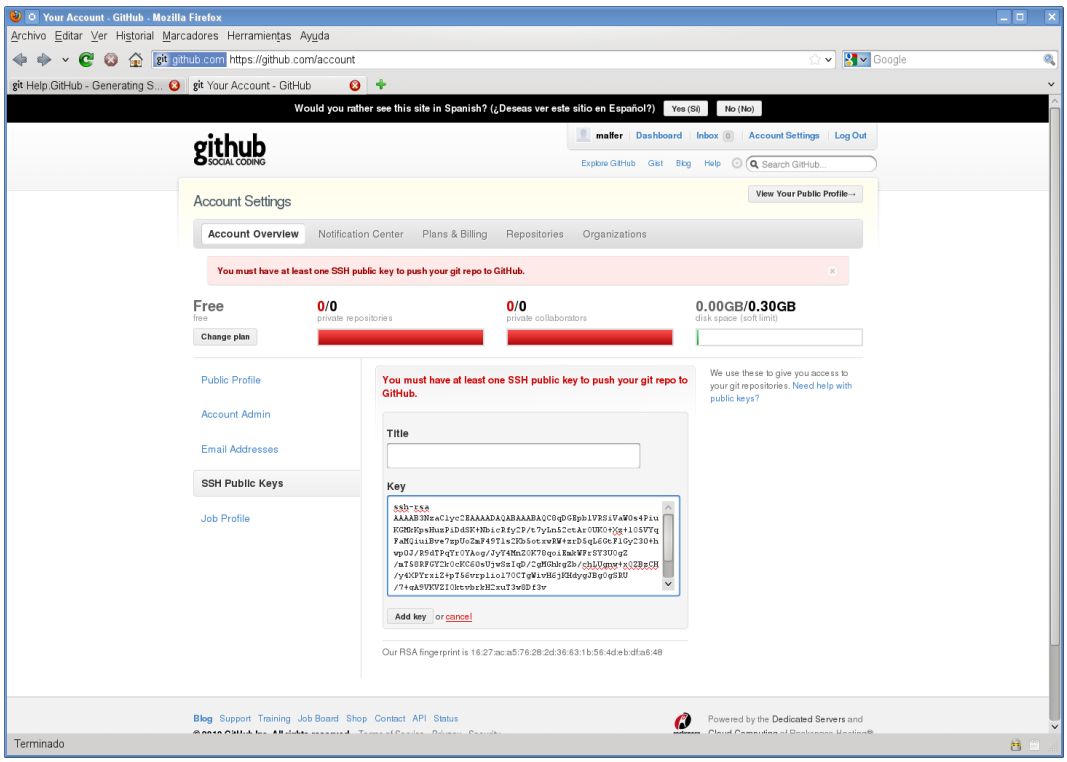
- Dispone de una página de ayuda fantástica sobre todos los aspectos de Git y del uso del servicio [29].
- Está perfectamente diseñado para realizar forks de un proyecto y enviar peticiones de pull al desarrollador principal, que es la mejor manera de trabajar colaborativamente con Git.
- El usuario que crea un repositorio es su único dueño y lo puede borrar si así lo quiere.
- Tiene prestaciones adicionales interesantes como crear una wiki o manejar reparos.

Registro en GitHub y creación del repositorio

Creación del usuario. Se introduce nombre, dirección de correo y contraseña.

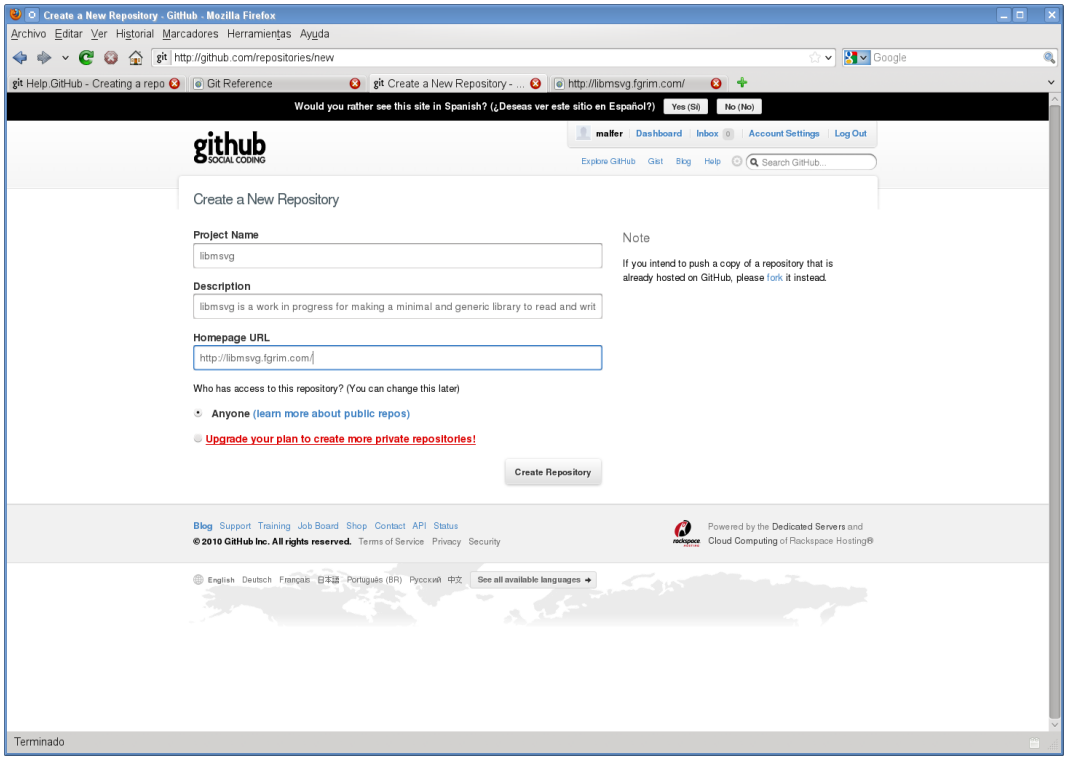


GitHub requiere crear una pareja de claves e introducir la clave pública para poder posteriormente subir los commit con una relación de confianza. Todo el proceso se explica detalladamente en la página de ayuda de GitHub.



Una vez introducida la clave se puede probar que funciona haciendo "ssh git@github.com". Aunque GitHub no provee acceso ssh, sí indica si la autentificación es correcta o no.

Creación del repositorio. Se introduce el nombre del proyecto, descripción, página web y que queremos que sea público.

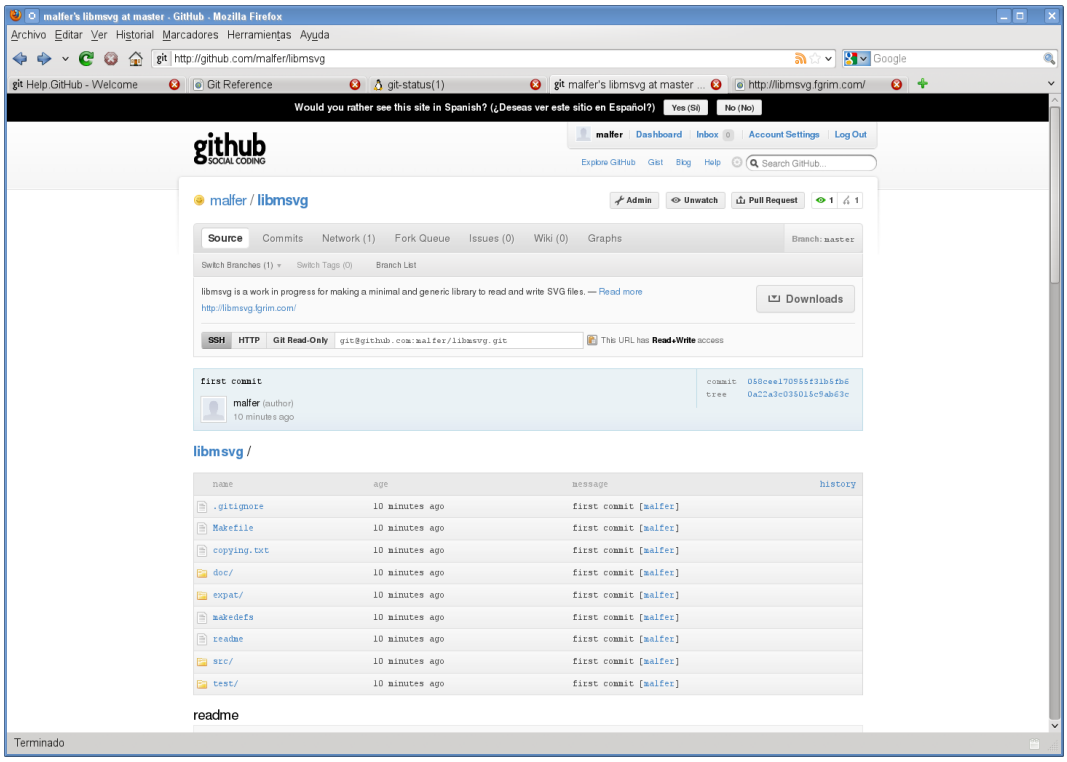


El primer push al repositorio público

El siguiente log muestra los comandos utilizados para realizar el primer push desde el repositorio local al repositorio público en GitHub:

```
mariano@zeravla5:~/libmsvg> git remote add origin git@github.com:malfer/libmsvg.git
                  <== Se asocia el nombre “origin” con el repositorio público”
mariano@zeravla5:~/libmsvg> git push origin master  <== El primer push
Enter passphrase for key '/home/mariano/.ssh/id_rsa':
Counting objects: 51, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (50/50), done.
Writing objects: 100% (51/51), 77.46 KiB, done.
Total 51 (delta 4), reused 0 (delta 0)
To git@github.com:malfer/libmsvg.git
 * [new branch]      master -> master
```

Y ahora ya podemos ir a http://github.com/malfer/libmsvg para ver nuestro repositorio público en toda su gloria:



Foro o lista de correo (tarea 2.4)

Para poder vehicular la participación de otros desarrolladores en la librería, es necesario disponer de un medio de comunicación, que puede ser un foro o una lista de correo cuyas conversaciones puedan archivarse y seguirse en línea.

He preferido utilizar un foro por ser más fácil de consultar y usar para personas que se acerquen al proyecto por primera vez. Además como es muy difícil generar interés inmediato por un nuevo proyecto de software libre hoy en día, el uso de listas de correo sin una comunidad real no tiene mucho sentido.

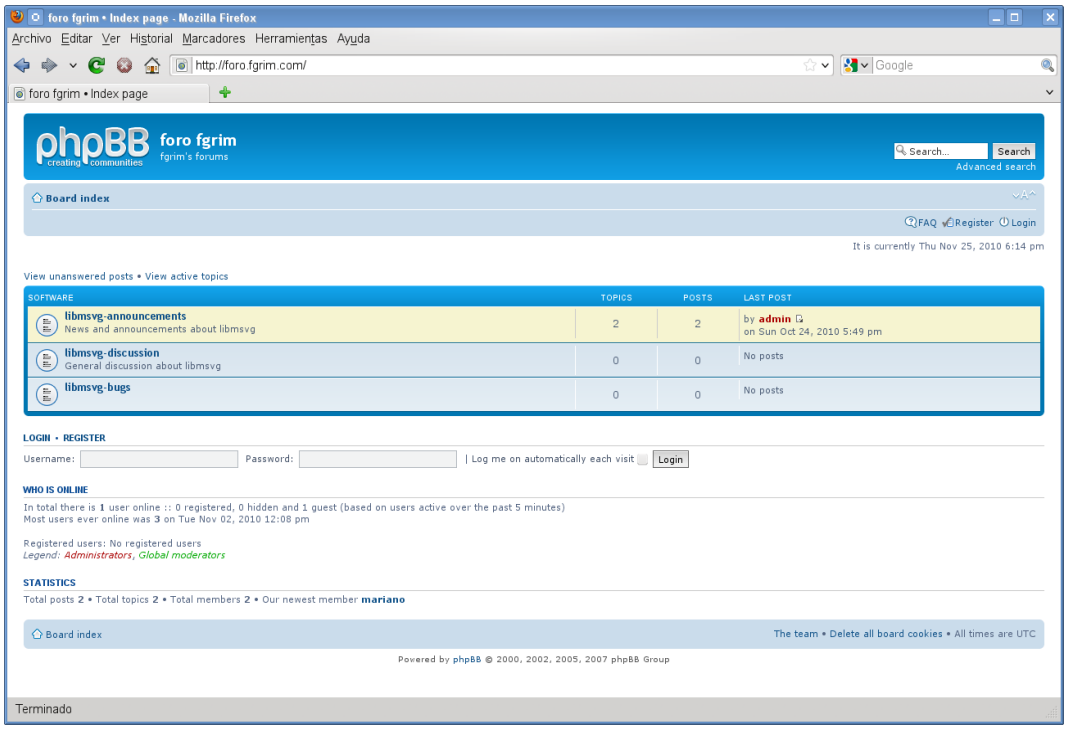
Instalación del foro

He elegido el software de foros por antonomasia phpBB [30] sobre todo porque instalarlo en mi dominio fue muy fácil, sólo tuve que acceder al panel de control de mi proveedor de hosting, elegir un subdominio (http://foro.fgrim.com), definir la clave del administrador y apretar un botón. La instalación desde cero tampoco es muy complicada, pero tienes que descargar el software, instalarlo y crear la base de datos.

Una vez instalado phpBB cree tres foros:

- **libmsvg-announcements**, que utilizaré para anuncios de nuevas versiones y otros eventos y en el que me he reservado derecho exclusivo de escritura.
- **libmsvg-discussion**, para discusión general y con acceso público.
- **libmsvg-bugs**, para reportar errores y con acceso público.

En la imagen siguiente puede verse el resultado final con dos temas creados en libmsvg-announcements, anunciando la primera versión de libmsvg y su disponibilidad en GitHub.



El idioma utilizado en las listas es el inglés para ampliar el público objetivo.

Aparición del SPAM

Hasta ahora, aunque veo que suben las lecturas de los mensajes que he dejado en la lista de anuncios, nadie ha dejado ningún mensaje relativo al proyecto. En cambio he tenido un éxito absoluto con el SPAM, no sólo dejan mensajes basura sino que crean montones de cuentas con identificativos raros aparentemente para nada. Casi cada día tengo que limpiar el foro.

Para ponérselo un poco más difícil a los spamers (y por desgracia un poco más complicado a quién quiera dejar un mensaje válido) activé la obligación de introducir un código captcha [31] y después la obligación de tener que confirmar el alta por correo. Ante mi sorpresa ninguna de las dos acciones evitaron el SPAM. He tenido que activar también la moderación para que al menos no aparezcan los mensajes basura, pero aún así siguen dejándolos y creando usuarios.

Hace unos días apareció en Barrapunto un comentario [32] que me dió la explicación. Existe un mercado de descifrado de captchas para spamers. Reclutan personas en países con del tercer mundo para descifrarlos y cobran a medio dólar los 1000 captchas descifrados. El ingenio humano para ganar dinero con lo que sea, y a costa de quien sea, no tiene límites.

Publicitar proyecto (tarea 2.5)

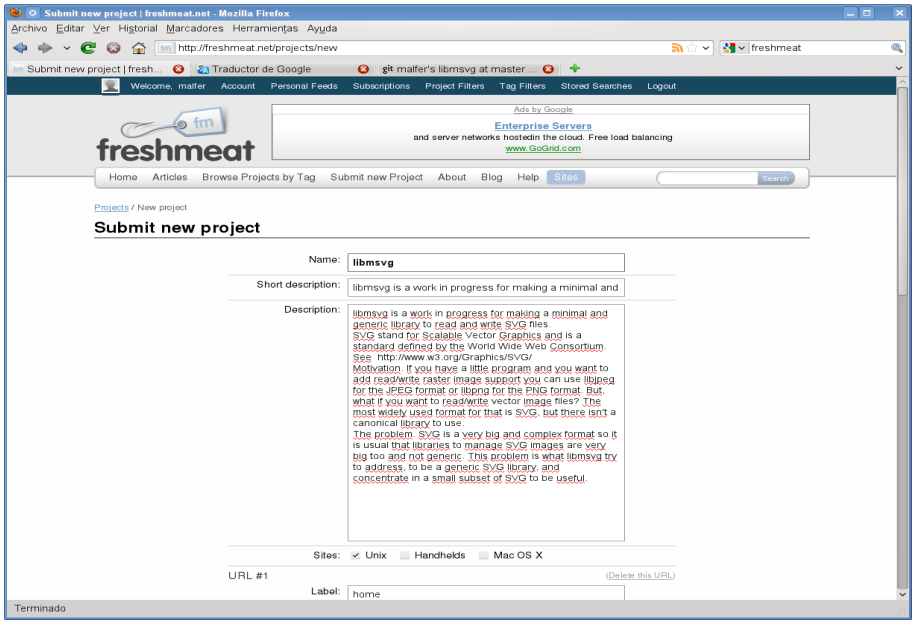
El objeto de esta tarea es dar a conocer la librería. He realizado cuatro actividades: registrar el proyecto en Freshmeat, anunciarlo en la lista de correo dedicada al formato SVG de la W3, en la lista de correo de GRX y en el grupo de Yahoo de desarrolladores de SVG.

Registro del proyecto en Freshmeat

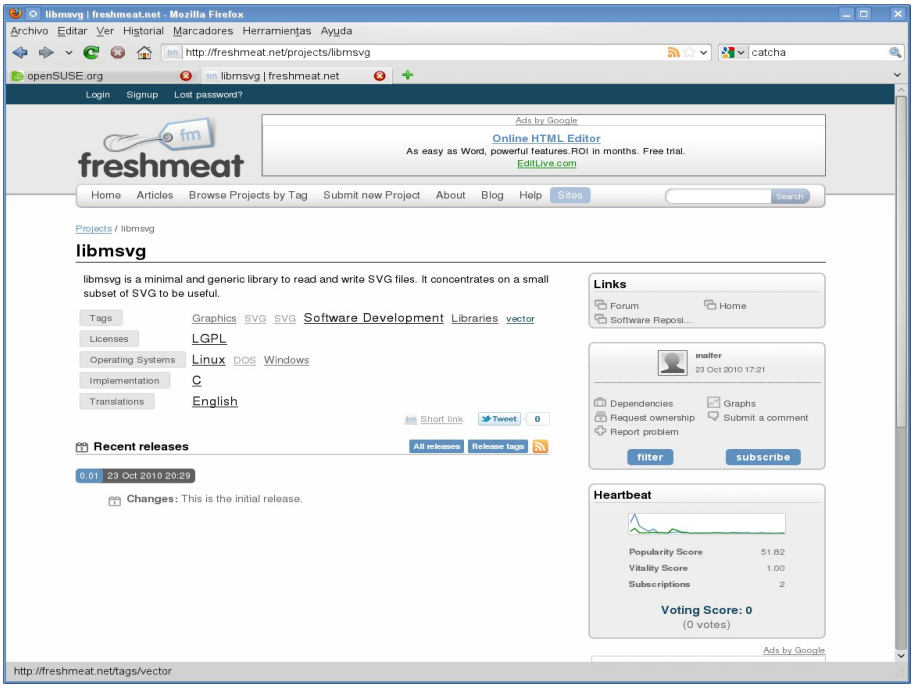
Freshmeat [21] es el canal de anuncios de software libre por antonomasia. Es el primer lugar a visitar si necesitas conocer que software libre existe de algún tema concreto.

Como ya tenía usuario en Freshmeat sólo tuve que iniciar sesión y pulsar en “Submit New Proyect”.

Aparece una ficha que hay que rellenar con el nombre del proyecto, descripción y datos como la página web y otros enlaces que consideres de interés. En la imagen de abajo se muestra parte de la ficha rellena.



Freshmeat es un servicio moderado, pasa uno o dos días hasta que te aprueban el proyecto. Tienen libertad para mejorar la redacción de la descripción, en mi caso redujeron considerablemente el "rollo" que metí en la mencionada descripción.



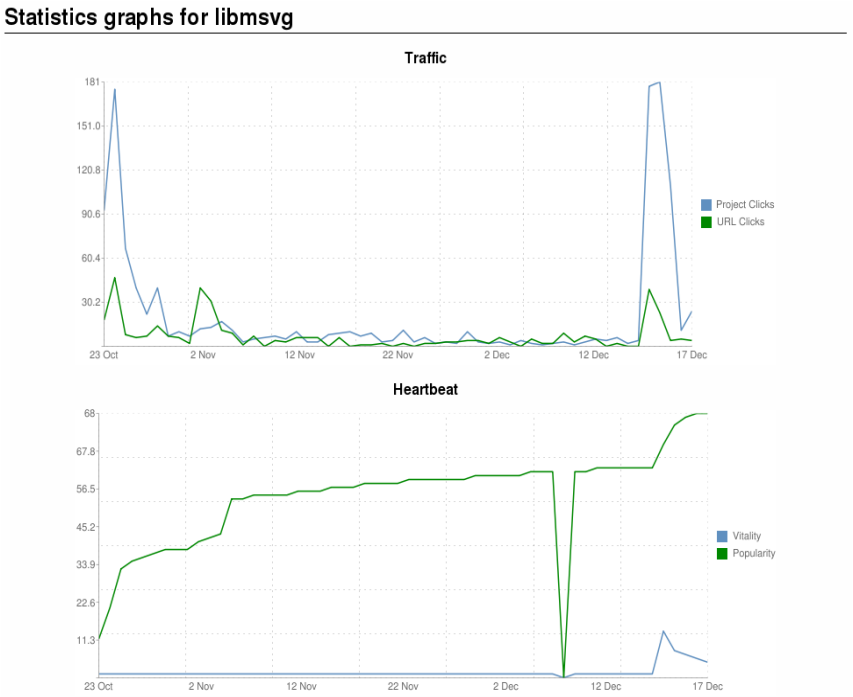
Una vez aprobado ya puedes anunciar la primera versión introduciéndola en la página del proyecto. En la imagen de arriba se muestra dicha página a 25 de

noviembre, unas semanas después del anuncio. Se ve en la gráfica que tuvo un pico de visitas tras el anuncio y luego bajó notablemente, también se ve que tiene dos subscriptores, que son personas que desean recibir un correo cuando se hagan nuevos anuncios del proyecto.

El efecto Freshmeat es considerable, antes del anuncio una búsqueda en Google sobre "libmsvg" devolvía 0 resultados. La misma búsqueda repetida a 26 de diciembre devuelve 4.500 resultados, la mayoría de webs de enlaces que replican el contenido de Freshmeat.

Realicé un segundo anuncio tras la liberación de la versión 0.02 de la librería.

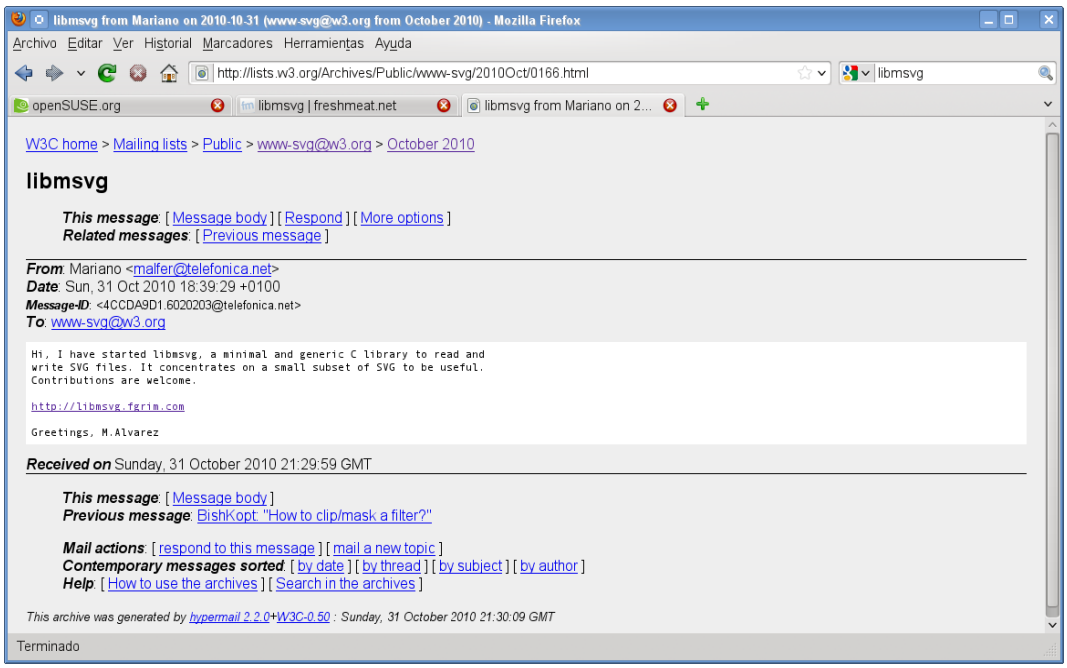
En el gráfico se puede ver el crecimiento de visitas resultante. También el aumento de los indicadores de vitalidad, que tiene que ver con el número de anuncios, y de popularidad, relacionado con el número de visitas y de subscriptores.



Anuncio en la lista de correo del W3

El World Wide Web Consortium [2] mantiene una lista de correo pública dedicada al formato SVG. No es necesario subscribirse para enviar un mensaje, aunque antes te piden en un correo que confirmes el permiso para que aparezca en el archivo público.

El archivo es accesible en http://lists.w3.org/Archives/Public/www-svg/ y mi mensaje, que muestro a continuación, en http://lists.w3.org/Archives/Public/www-svg/2010Oct/0166.html



Recibí dos respuestas, ambas indicándome amablemente que la lista era más bien para la discusión del formato SVG en sí. Uno de ellos me indicó como más adecuado anunciarlo en un grupo de discusión de Yahoo. Otro me pidió más detalles sobre el proyecto e intercambiamos algunos correos privados.

Anuncio en la lista de correo de GRX

GRX [9] es una librería de gráficos 2D de la que se deriva MGRX [10], que mantengo desde hace unos años y que es compatible, hasta cierto punto, con la original.

GRX dispone de una lista de correo no muy activa hoy en día que está archivada en http://www2.grx.gnu.de/archive/grx/en/

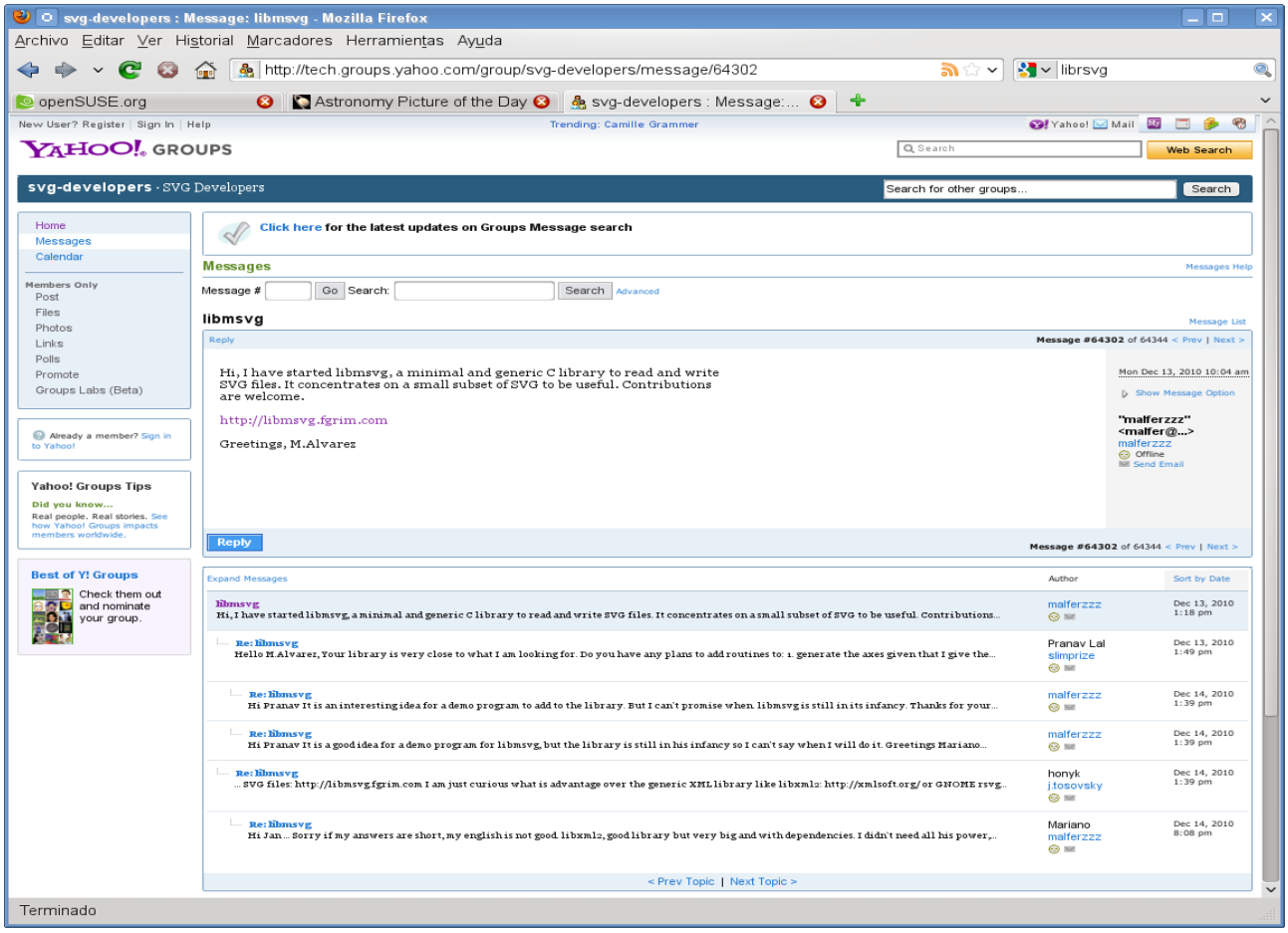
Como planeo hacer una interfaz de MGRX/GRX con libmsvg envié un anuncio a dicha lista, en la que aún estoy suscrito y que puede consulatrse en ésta dirección: http://www2.grx.gnu.de/archive/grx/en/mail608.html

No he recibido respuesta a éste anuncio.

Anuncio en el grupo de Yahoo svg-develpers

Tras la recomendación recibida en la lista del W3 me uní al grupo de Yahoo svg-developers (http://tech.groups.yahoo.com/group/svg-developers/) y realicé un anuncio.

Recibí algunas respuestas en el grupo y en privado interesándose por la librería. En la imagen se muestra el anuncio y el árbol de respuestas.



Documentación final (tarea 2.6)

Uno de los puntos débiles de la mayoría de proyectos de software libre es la documentación, que suele ser escasa y muchas veces desactualizada.

Muchos proyectos se decantan por documentación semiautomática generada a partir de los fuentes, con lo que consiguen un documento de referencias, un catálogo de funciones y estructuras de datos. Siendo éste interesante para un desarrollador que ya utilice el software, creo que el documento más importante para alguien que echa un primer vistazo al proyecto es un buen tutorial. Un documento que explique que puede hacerse con la librería y como se hace. Este es el sentido que he dado a la guía del programador que he construido y cuya versión final incluyo como Anexo 2. Está escrito directamente en html y en idioma inglés.

El otro documento importante es el README del proyecto que incluye las instrucciones de instalación. Lo incluyo como Anexo 1.

Capítulo 5. Fase de mantenimiento y mejora

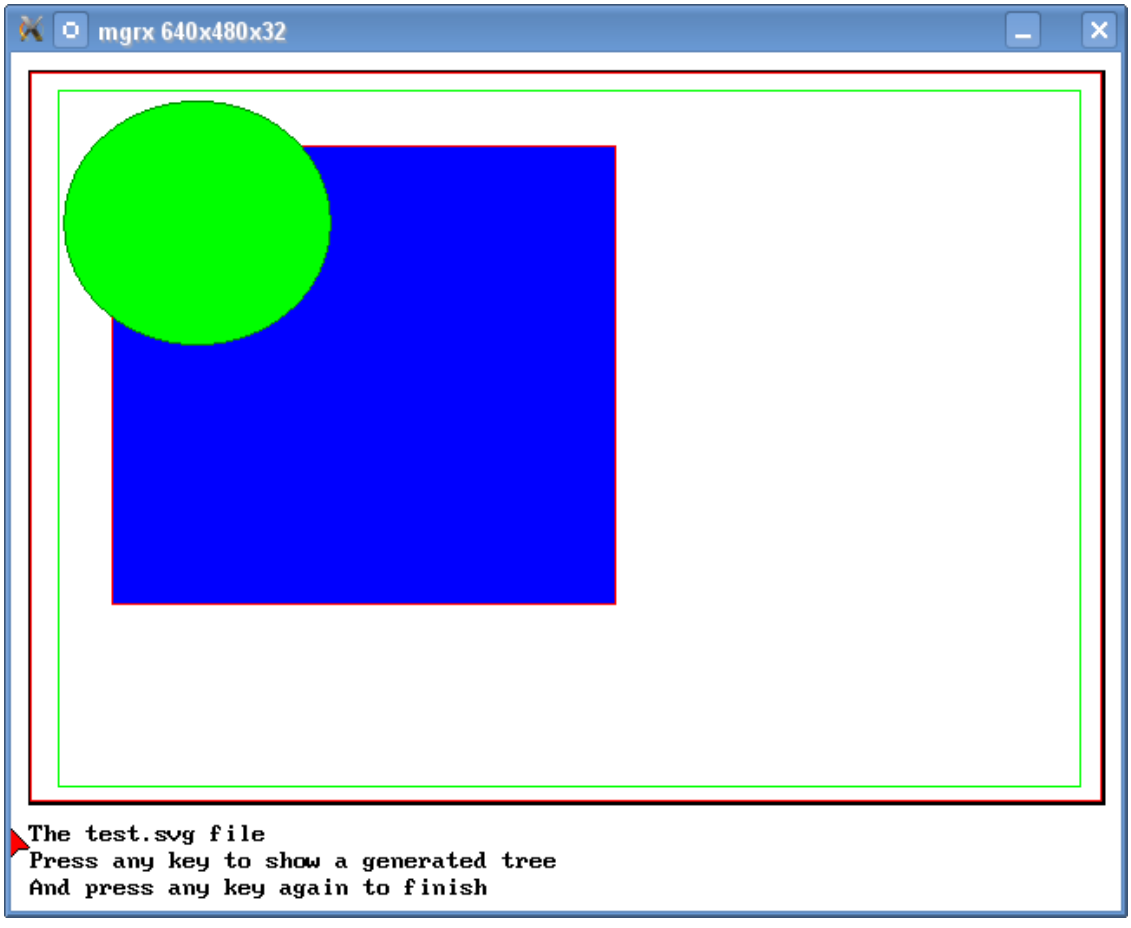
Esta fase arrancó tras la finalización de la fase de desarrollo y en paralelo a la fase de publicación.

Es necesario seguir mejorando la librería e incluir posibles aportaciones de terceros. Esto generará nuevas noticias para actualizar la web y enviar mensajes sobre los canales de información del proyecto.

Desde la publicación de la primera versión de la librería he continuado el desarrollo y realizado 14 commints adicionales.

Fruto de este trabajo es la versión 0.02 de la librería, publicada en la página web y anunciada en Freshmeat. Esta segunda versión ya es capaz de tratar árboles del tipo COOKED.

Tras la publicación de la segunda versión he añadido un pequeño programa de ejemplo capaz ya de renderizar cículos y cuadrados utilizando libmsvg y la librería gráfica mgrx. Los cambios están en GitHub.



Capítulo 6. Conclusiones

Al estar encuadrado este trabajo de fin de Máster en el área de desarrollo de aplicaciones de software libre, todo el trabajo gira en demostrar competencias tanto en el desarrollo de software libre como en su liberación como proyecto real a la comunidad utilizando los medios habituales de este tipo de proyectos.

Actualmente el software libre ya no es un fenómeno minoritario y he podido realizar el desarrollo con un sistema operativo libre y utilizando un compilador y herramientas libres. He podido además construir el núcleo de libmsvg a partir de otra librería libre e incluso he escrito esta memoria utilizando un procesador de texto libre.

Igualmente para la liberación y publicación del proyecto me he beneficiado de abundantes medios disponibles para que los desarrolladores de software libre pongan su trabajo a disposición de otros e invitarlos a participar.

Considero que he alcanzado los objetivos que me había propuesto tanto de desarrollo como de publicación del proyecto, aunque no haya conseguido, de momento, crear un grupo en torno a libmsvg.

Existe una imagen idealizada de la comunidad de software libre como un conjunto compacto y voluntarioso de programadores dispuestos a echar una mano y tirar de cada proyecto que alguien pueda imaginar y presentar.

Obviamente no es así. Por una parte tenemos un grupo de programadores profesionales, subvencionados por empresas privadas, que producen software libre en áreas de interés para dichas empresas. Estos son los menos. Tenemos también a los que, bien por genialidad, bien por suerte, han conseguido que su aportación sea ampliamente utilizada y reconocida, estos tampoco son muchos.

Por otra parte tenemos a la gran mayoría de desarrolladores de software libre que utiliza su tiempo libre en esta actividad. Para ellos es una afición al margen de aquello que les sirve para ganarse la vida. No es de extrañar que su empeño y entusiasmo vaya dirigido a aquellas áreas que más le gratifiquen intelectualmente y que además ese entusiasmo no sea constante sino que sufra altibajos según su situación laboral, personal y familiar.

En este marco no me sorprende en absoluto que mi pequeño proyecto no haya creado un grupo de programadores a su alrededor de forma inmediata, ya lo suponía. Sólo en GiHub hay millón y medio de repositorios con más de medio millón de

programadores. Tocan a 3 proyectos cada uno. Y es que lo más motivante para un programador de software libre es dirigir su propio proyecto.

Esto no es óbice para que la gente colabore en proyectos ajenos bien cuando está aprendiendo (y suelen elegir proyectos grandes con proyección), bien porque usan el producto (y dan el salto de usuario a desarrollador) o bien porque necesitan piezas para su propio proyecto. Aquí libmsvg está bien situado y tengo puestas esperanzas en él, las barreras de entrada son pequeñas y creo que puede ser de utilidad para otras iniciativas (incluida una de las mías: mgrx).

No obstante a libmsvg aún le queda un largo trecho para ser una pieza de software acabada. Uno de los correos que he recibido me preguntaba si estaba suficientemente maduro para incluirlo en su propio proyecto. Le conteste sinceramente que no, pero que (haciéndole el típico guiño ;-) me podía ayudar a ello.

En cualquier caso y en la medida de mis fuerzas y tiempo disponible, pretendo seguir desarrollando libmsvg. Era de hecho una idea que ya tenía antes de emprender este Máster. Máster que me ha permitido dar forma a las primeras versiones de la librería y darla a conocer.

Referencias

[1] Descripción del formato SVG: http://www.w3.org/Graphics/SVG/

[2] Página web del World Wide Web Consortium: http://www.w3.org

[3] Definición de software libre: http://www.gnu.org/philosophy/free-sw.es.html

[4] Licencia LGPL v2: http://www.gnu.org/licenses/old-licenses/lgpl-2.0.html

[5] Página web del compilador DJGPP: http://www.delorie.com/djgpp

[6] Proyecto netpbm: http://netpbm.sourceforge.net/

[7] Librería JPEG: http://www.ijg.org/

[8] Librería PNG: http://www.libpng.org/pub/png/libpng.html

[9] Librería GRX: http://grx.gnu.de

[10] Librería MGRX: http://mgrx.fgrim.com

[11] Librería librsvg: http://librsvg.sourceforge.net/

[12] Cairo, la librería de gráficos 2D de GNOME: http://cairographics.org/

[13] Especificación SVG Tiny 1.2: http://www.w3.org/TR/SVGTiny12/

[14] Librería libexpat: http://expat.sourceforge.net/

[15] Librería libxml2: http://xmlsoft.org/

[16] Librería tinyxml: http://www.grinninglizard.com/tinyxml/

[17] Librería libsvg: http://cairographics.org/snapshots/

[18] Desarrollo de proyectos de software libre, publicado por la UOC (XP06/M2120/02157)

[19] Página web de SourceForge: http://sourceforge.net/

[20] Página web del sistema de control de versiones Git: http://git-scm.com/

[21] Página web de Freshmeat: http://freshmeat.net/

[22] Herramienta de gestión de proyectos GanttProject: http://www.ganttproject.biz/

[23] Transformaciones en SVG:
http://www.w3.org/TR/SVGTiny12/coords.html#EstablishingANewUserSpace

[24] TestSuite de imágenes SVG: http://www.w3.org/Graphics/SVG/Test/20070907/

[25] Tutorial de libexpat: http://www.xml.com/pub/a/1999/09/expat/index.html

[26] Linus Torvald, creador de Linux y de Git:
http://es.wikipedia.org/wiki/Linus_Torvalds

[27] Tech Talk de Linus Torvalds en Google sobre Git:
http://www.youtube.com/watch?v=4XpnKHJAok8

[28] Página web de GitHub: http://github.com/

[29] Página de ayuda de GitHub: http://help.github.com/

[30] Página web de PhpBB: http://www.phpbb.com/

[31] Qué es un captcha: http://es.wikipedia.org/wiki/Captcha

[32] Sobre descifrado de captchas por spammers en Barrapunto:
http://barrapunto.com/articles/10/11/25/0832240.shtml

Anexo 1. Fichero readme

```
Welcome to libmsvg version 0.02

libmsvg is a work in progress to make a minimal and generic library to read
and write SVG files.

SVG stand for Scalable Vector Graphics and is a standard defined by the World
Wide Web Consortium. See http://www.w3.org/Graphics/SVG/

To be useful, libmsvg concentrates in a small subset of the SVG Tiny 1.2
specification.

Supported platforms
===================

libmsvg is programmed in ANSI-C, so it must compile in every plattform, you
only have to build the makefiles. We provide the makefiles for these three
supported platforms:

  Linux using the gcc compiler and gmake
  DOS using the DJGPP compiler and gmake
  Win32 using the Mingw compiler and gmake

Dependencies
============

libmsvg only depends of libexpat 1.2 (http://www.jclark.com/xml/expat.html),
but we include our own copy of the expat library, so there are no depndencies
at all.

libmsvg installation instructions
=================================

Requirements:
-------------

  The source files:  libmsvg0002.tar.gz or msvg0002.zip
  This document:     readme

A. Unpacking the libmsvg archive
--------------------------------

  1) Choose and download the .tar.gz or .zip package. You can use either.
     Usually Linux users choose the .tar.gz and DJGPP/Mingw users choose
     the .zip.

  2) 'cd' to a directory where you want the libmsvg file tree to be created
     as a subdirectory. Examples are:

       DJGPP : cd C:\DJGPP\contrib
       Mingw : cd C:\MINGW\contrib
       Linux : cd /usr/local/src

  3) unpack the libmsvg archive:

          tar xzvf libmsvg0001.tar.gz
     or
```

```
         unzip msvg0001.zip

     This will create the subdirectory 'libmsvg'.

B. Compiling libmsvg
--------------------

  1) Go to libmsvg base dir.

  2) Edit the "makedefs" file and set to 'y' only one of this variables:

     LINUX_VERSION
     DJGPP_VERSION
     MINGW_VERSION

     By default the LINUX_VERSION is checked

  3) Run 'make' ('mingw32-make' for Mingw users)

     Note for DJGPP/Mingw users: Do _not_ use an environment variable
     `SHELL' leading to `bash', e.g. `SHELL=/djgpp/bin/bash.exe'.
     Some parts of the DJGPP/Mingw Makefiles require `command.com'.

C. Testing libmsvg
------------------

  1) Go to the 'test' subdir.

  2) run the test programs.


D. Installing libmsvg
---------------------

  Copy the library 'libmsvg.a' from 'src' subdir to the system library
  directory.

  Copy the header file 'msvg.h' from 'src' subdir to your system
  include directory.

  Or you can let makefiles do it for you, running 'make install'.

  You can uninstall the library running 'make uninstall'.

  Note for Linux users: probably you must be root to do that.

Help
====

  Read the programmer's guide in the 'doc' subdir.

  Check the libmsvg site (libmsvg.fgrim.com) for updates, tips, ...

  Send me a mail <malfer at telefonica.net>

License
=======
  libmsvg is free software. You can redistribute it and/or modify it under
  the LGPL license. See the 'copying.txt' file for details.

Enjoy, Mariano Alvarez <malfer at telefonica.net>
```

Anexo 2. Manual del programador

libmsvg 0.02 Programmer's guide

Last update: December 7, 2010

Abstract

libmsvg is a minimal and generic C library to read and write SVG files.

SVG stand for Scalable Vector Graphics and is a standard defined by the World Wide Web Consortium (see http://www.w3.org/Graphics/SVG/).

libmsvg concentrates on a small subset of SVG to be useful. More specifically on a subset of the SVG Tiny 1.2 specification. The subset is described in Appendix A.

Contents

- Starting with examples
- The MsvgElement tree
- The MsvgElement structure
- Reading SVG files
- Building a RAW MsvgElement tree by program
- Building a COOKED MsvgElement tree by program
- Manipulating a MsvgElement tree
- Writing SVG files
- Appendix A, the libmsvg SVG subset

Starting with examples

I think the best way to start with a programmer's guide is to show examples, so here there are.

Example 1

This example read a SVG file in a MsvgElement tree

```
#include <stdio.h>
#include "msvg.h"

int main(int argc, char **argv)
{
  MsvgElement *root;

  if (argc <2)
    return 0;

  root = MsvgReadSvgFile(argv[1]);

  if (root == NULL) {
```

```
    printf("Error opening %s\n", argv[1]);
    return 0;
  }

  /* Now you can process the structure. By example: */
  MsvgPrintElementTree(stdout, root, 0);

  return 1;
}
```

Example 2

This example builds a MsvgElement tree and writes it to a file

```
#include <stdio.h>
#include "msvg.h"

#define TESTFILE "msvgt2.svg"

int main(int argc, char **argv)
{
  MsvgElement *root, *son;

  root = MsvgNewElement(EID_SVG, NULL);
  MsvgAddAttribute(root, "version", "1.2");
  MsvgAddAttribute(root, "baseProfile", "tiny");
  MsvgAddAttribute(root, "viewBox", "0 0 400 400");

  son = MsvgNewElement(EID_RECT, root);
  MsvgAddAttribute(son, "x", "1");
  MsvgAddAttribute(son, "y", "1");
  MsvgAddAttribute(son, "width", "398");
  MsvgAddAttribute(son, "height", "398");
  MsvgAddAttribute(son, "stroke", "#F00");
  MsvgAddAttribute(son, "fill", "#FFF");

  son = MsvgNewElement(EID_RECT, root);
  MsvgAddAttribute(son, "x", "11");
  MsvgAddAttribute(son, "y", "11");
  MsvgAddAttribute(son, "width", "380");
  MsvgAddAttribute(son, "height", "380");
  MsvgAddAttribute(son, "stroke", "#0F0");
  MsvgAddAttribute(son, "fill", "none");
  MsvgPrintElementTree(stdout, root, 0);

  if (!MsvgWriteSvgFile(root, TESTFILE)) {
    printf("Error writing %s\n", TESTFILE);
    return 0;
  }

  return 1;
}
```

How to compile the example programs

We assume here that you have the libmsvg library previously installed. If not, please read the "readme" file for installation instructions.

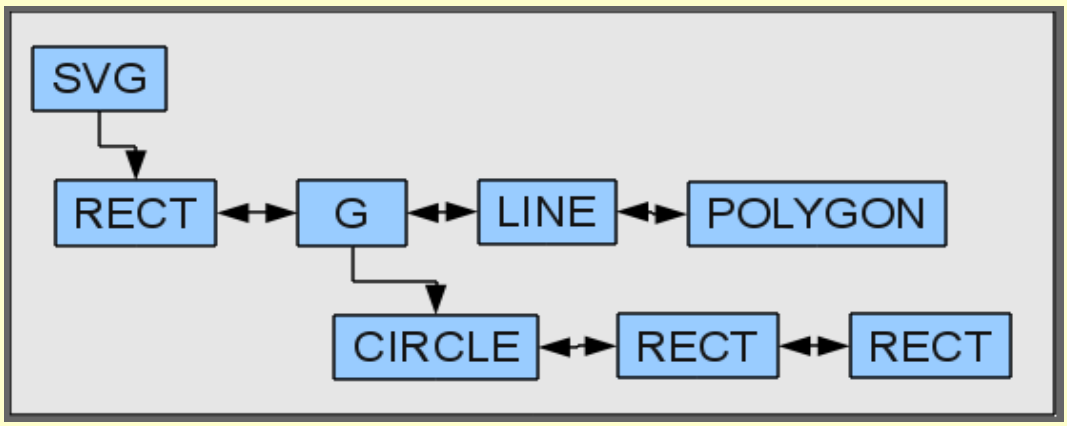
These are the command lines to compile the examples for the three supported platforms:

- Linux: gcc -o example example.c -lmsvg
- DJGPP: gcc -o example.exe example.c -lmsvg
- Mingw: gcc -o example.exe example.c -lmsvg -mconsole

The MsvgElement tree

The central structure of libmsvg is the MsvgElement tree. Every MsvgElement has a defined element id (eid), a pointer to his father, pointers to his previous and next siblings and a pointer to his first son. The root element must be a EID_SVG. In this version of libmsvg only EID_SVG and EID_G can have son elements. The other supported elements are EID_RECT, EID_CIRCLE, EID_ELLIPSE, EID_LINE, EID_POLYLINE and EID_POLYGON.

This graph represents an example of a MsvgElement tree:



Every MsvgElement can have attributes, they can be of two types: generic or specific. The type of MsvgElement attributes in a tree is determined by a variable in the root element: root->psvgattr->tree_type. If this variable is RAW_SVGTREE all attributes are generic. Generic attributes are simple key,value pairs. If it is COOKED_SVGTREE all attributes are specific. Specific attributes are typed variables specific to each element id type.

The RAW_SVGTREE tree type

After a SVG file is loaded to a MsvgElement tree by the MsvgReadSvgFile function it is marked as a RAW_SVGTREE. Only the supported elements are inserted in the tree, but all the read attributes are stored like generic attributes. This stage of the tree can suffice for some programs, elements and attributes can be added, deleted or reordered and finally be written to a file using the MsvgWriteSvgFile function.

The COOKED_SVGTREE tree type

Using the MsvgRaw2CookedTree funtion you can convert a MsvgElement tree to the COOKED_SVGTREE type. Only the supported attributes are processed and converted to specific attributes. In this state the tree is easier to be manipulated by a program.

The FRIED_SVGTREE tree type

A third tree type is planned, using another planned function (to be added to the library) you can convert a tree from COOKED_SVGTREE to FRIED_SVGTREE. In this state, all the elements are serialized, resolving each posible EID_G element, and all coordinates are truncated to integer values. This will be the ideal state for drawing a MsvgElemet tree using a graphics package.

The MsvgElement structure

After reading the previous section it must be easy to undestand the MsvgElement structure from the msvg.h include file:

```
typedef struct _MsvgElement *MsvgElementPtr;

typedef struct _MsvgElement {
  enum EID eid;
  MsvgElementPtr father; /* pointer to father element */
  MsvgElementPtr psibling; /* pointer to previous sibling element */
  MsvgElementPtr nsibling; /* pointer to next sibling element */
  MsvgElementPtr fson; /* pointer to first son element */
  MsvgAttributePtr fattr; /* pointer to first generic attribute */
  union { /* specific attributes */
    MsvgSvgAttributes *psvgattr;
    MsvgGAttributes *pgattr;
    MsvgRectAttributes *prectattr;
    MsvgCircleAttributes *pcircleattr;
    MsvgEllipseAttributes *pellipseattr;
    MsvgLineAttributes *plineattr;
    MsvgPolylineAttributes *ppolylineattr;
    MsvgPolygonAttributes *ppolygonattr;
  };
} MsvgElement;
```

Generic attributes are stored in a simple linked list of MsvgAttribute variables:

```
typedef struct _MsvgAttribute *MsvgAttributePtr;

typedef struct _MsvgAttribute {
  char *key; /* key attribute */
  char *value; /* value attribute */
  MsvgAttributePtr nattr; /* pointer to next attribute */
} MsvgAttribute;
```

Specific attributes are different for each element id and stored in a union, you can inspect each one in the msvg.h include file.

Reading SVG files

Using the MsvgReadSvgFile function you load a SVG file in a MsvgElement tree.

```
  MsvgElement *root;
  root = MsvgReadSvgFile("filein.svg");
```

If the file doesn't exists or it isn't a valid SVG file root will be NULL, so you must check it. Only

the supported elements are stored in the tree, the not supported ones are silently ignored. The tree will be a RAW tree, with all the element attributes stored as generic attributes. If you want a COOKED tree you can use the MsvgRaw2CookedTree funtion:

```
int result;
result = MsvgRaw2CookedTree(root);
```

result will be true if all was right, false otherwise

Building a RAW MsvgElement tree by program

Using only two function we can cosntruct a MsvgElement tree by program. The MsvgNewElement function takes two parameters: the element id and the father element, that can be NULL. It returns the pointer to the constructed element.

So we begin constructing the SVG element passing NULL in the father parameter, because it is the root element. By default the tree will be RAW_SVGTREE.

```
MsvgElement *root;
root = MsvgNewElement(EID_SVG, NULL);
```

Now, we add an attribute to set the drawing limits using the MsvgAddAttribute function.

```
MsvgAddAttribute(root, "viewBox", "0 0 400 400");
```

We continue adding two son elements, a RECT element and a G element.

```
MsvgElement *son;
son = MsvgNewElement(EID_RECT, root);
MsvgAddAttribute(son, "x", "1");
MsvgAddAttribute(son, "y", "1");
MsvgAddAttribute(son, "width", "398");
MsvgAddAttribute(son, "height", "398");
MsvgAddAttribute(son, "stroke", "#F00");
MsvgAddAttribute(son, "fill", "#FFF");
son = MsvgNewElement(EID_G, root);
MsvgAddAttribute(son, "stroke", "#0F0");
MsvgAddAttribute(son, "fill", "none");
MsvgPrintElementTree(stdout, root, 0);
```

Finally we add two son CIRCLE elements to the G element. Note that they inherit the stroke and fill attributes.

```
MsvgElement *soon2;
son2 = MsvgNewElement(EID_CIRCLE, son);
MsvgAddAttribute(son2, "cx", "100");
MsvgAddAttribute(son2, "cy", "200");
MsvgAddAttribute(son2, "r", "80");
son2 = MsvgNewElement(EID_CIRCLE, son);
MsvgAddAttribute(son2, "cx", "300");
MsvgAddAttribute(son2, "cy", "200");
MsvgAddAttribute(son2, "r", "80");
```

We have now our MsvgElement tree and we can manipulate it or write it to a file.

Building a COOKED MsvgElement tree by program

Constructing a COOKED tree is the same like constructing a RAW one, except we don't use the MsvgAddAttribute function. Instead we set directly the element variables, there are no functions to hide the variables, because we are programmers and we know what are we doing, doesn't it.

```
MsvgElement *root;
root = MsvgNewElement(EID_SVG, NULL);
root->psvgattr->vb_min_x = 0;
root->psvgattr->vb_min_y = 0;
root->psvgattr->vb_width = 400;
root->psvgattr->vb_height = 400;
root->psvgattr->tree_type = COOKED_SVGTREE;

MsvgElement *son;
son = MsvgNewElement(EID_RECT, root);
son->prectattr->x = 1;
son->prectattr->y = 1;
son->prectattr->width = 398;
son->prectattr->height = 398;
son->prectattr->fill_color = 0XFFFFFF;
son->prectattr->stroke_color = 0XFF0000;
son = MsvgNewElement(EID_G, root);
son->gattr->fill_color = NO_COLOR;
son->gattr->stroke_color = 0X00FF00;

MsvgElement *soon2;
son2 = MsvgNewElement(EID_CIRCLE, son);
son2->pcircleattr->cx = 100;
son2->pcircleattr->cy = 200;
son2->pcircleattr->r = 80;
son2 = MsvgNewElement(EID_CIRCLE, son);
son2->pcircleattr->cx = 300;
son2->pcircleattr->cy = 200;
son2->pcircleattr->r = 80;
```

Manipulating a MsvgElement tree

There are some functions to manipulate a MsvgElement tree.

```
void MsvgPruneElement(MsvgElement *el);
```

The MsvgPruneElement function prune an element from his tree and them we can insert it in another tree or in another point of the same tree. Note that if the pruned element has sons, it retains them after pruned.

```
void MsvgDeleteElement(MsvgElement *el);
```

The MsvgDeleteElement does two things, prunes the element from his tree and deletes it, freeing the allocated memory used. Note that if the deleted element has sons, they are deleted too.

```
int MsvgInsertSonElement(MsvgElement *el, MsvgElement *father);
int MsvgInsertPSiblingElement(MsvgElement *el, MsvgElement *sibling);
int MsvgInsertNSiblingElement(MsvgElement *el, MsvgElement *sibling);
```

This three functios insert an element (that can be a subtree if it has dons) in the desired point of a tree. The MsvgInsertSonElement fucntion inserts the element like the last son of the indicated

father. The MsvgInsertPSiblingElement function inserts the element like a previous sibling to the indicated sibling. And the MsvgInsertNSiblingElement function inserts the element like a next sibling. The three functions return 1 if all was ok, 0 otherwise.

Writing SVG files

Using the MsvgWriteSvgFile function you can write a MsvgElement tree to a file.

```
int MsvgWriteSvgFile(root, "fileout.svg")
```

MsvgWriteSvgFile only know how to write RAW trees. If you want to write a COOKED tree you need to use the MsvgCooked2RawTree funtion (to be added to the library) first:

```
int result;
result = MsvgCooked2RawTree(root);
```

Appendix A, the libmsvg SVG subset

This table lists the SVG elementes and attributes that will be supported when libmsvg reaches version 0.1.

Each row lists an element, the supported specific attributes and the possible elements that may be sons.

Note that specific attributes only affects to COOKED trees, RAW trees can have any attribute.

A status mark is added to each attribute: (ok) means it is already supported, (TO DO) means... you know: to do.

Element	Specific supported attributes (status)	Possible son elements
<svg>	viewbox="min-x min-y width height" (ok) width="width" (ok) height="height" (ok) viewport-fill="color" (ok) viewport-fill-opacity="n" (ok) version="valor" (TO DO) baseprofile="valor" (TO DO) preserveaspectratio="valor" (TO DO)	<g>, <rect>, <circle>, <ellipse>, <line>, <polyline>, <polygon>
<g>	id="valor" (ok) transform="translate(x,y)" (TO DO)	<g>, <rect>, <circle>, <ellipse>, <line>, <polyline>, <polygon>

	transform="rotate(angulo)" (TO DO) transform="scale(factor)" (TO DO) fill="color" (ok) fill-opacity="n" (ok) stroke="color" (ok) stroke-width="n" (ok) stroke-opacity="n" (ok)	
<rect>	x="n" (ok) y="n" (ok) width="n" (ok) height="n" (ok) rx="n" (ok) ry="n" (ok) id="valor" (ok) transform="translate(x,y)" (TO DO) transform="rotate(angulo)" (TO DO) transform="scale(factor)" (TO DO) fill="color" (ok) fill-opacity="n" (ok) stroke="color" (ok) stroke-width="n" (ok) stroke-opacity="n" (ok)	
<circle>	cx="n" (ok) cy="n" (ok) r="n" (ok) id="valor" (ok) transform="translate(x,y)" (TO DO) transform="rotate(angulo)" (TO DO)	

	transform="scale(factor)" (TO DO) fill="color" (ok) fill-opacity="n" (ok) stroke="color" (ok) stroke-width="n" (ok) stroke-opacity="n" (ok)	
<ellipse>	cx="n" (ok) cy="n" (ok) rx="n" (ok) ry="n" (ok) id="valor" (ok) transform="translate(x,y)" (TO DO) transform="rotate(angulo)" (TO DO) transform="scale(factor)" (TO DO) fill="color" (ok) fill-opacity="n" (ok) stroke="color" (ok) stroke-width="n" (ok) stroke-opacity="n" (ok)	
<line>	x1="n" (ok) y1="n" (ok) x2="n" (ok) y2="n" (ok) id="valor" (ok) transform="translate(x,y)" (TO DO) transform="rotate(angulo)" (TO DO) transform="scale(factor)" (TO DO) stroke="color" (ok)	

	stroke-width="n" (ok) stroke-opacity="n" (ok)	
<polyline>	points="data" (ok) id="valor" (ok) transform="translate(x,y)" (TO DO) transform="rotate(angulo)" (TO DO) transform="scale(factor)" (TO DO) stroke="color" (ok) stroke-width="n" (ok) stroke-opacity="n" (ok)	
<polygon>	points="data" (ok) id="valor" (ok) transform="translate(x,y)" (TO DO) transform="rotate(angulo)" (TO DO) transform="scale(factor)" (TO DO) fill="color" (ok) fill-opacity="n" (ok) stroke="color" (ok) stroke-width="n" (ok) stroke-opacity="n" (ok)	