Inside Cazabe

Autor
Darien Alonso Camacho
dalonso09@graduados.uci.cu

"Contemple desecharlo; de todos modos tendrá que hacerlo". Diciéndolo de otro modo: no se entiende cabalmente un problema hasta que se implementa la primera solución. La siguiente vez quizás uno ya sepa lo suficiente para solucionarlo. Así que si quieres resolverlo, disponte a empezar de nuevo al menos una vez.
Eric S. Raymond, La Catedral y el Bazar.

Introducción

El proyecto Cazabe persigue desarrollar una herramienta que permita crear multimedias y animaciones vectoriales en ambientes libres, sustituyendo a Macromedia/Adobe Flash.

El presente documento se elabora con el objetivo de arrojar luz sobre el funcionamiento interno del proyecto; es una guía para entender el código fuente y la interacción entre clases y objetos. Es apropiado para desarrolladores, no para usuarios finales. Los diagramas incluidos aquí, que contienen relaciones entre clases, interacciones de objetos, etc., no están escritos en UML, solo son imágenes representativas, permitiendo que desconocedores de este lenguaje de modelado también puedan informarse.

Los interesados en comprender a cabalidad los pormenores de implementación deben revisar, además, los comentarios hechos en el propio código fuente, debido a que constituyen un importante complemento a este manual.

Quien desee agregar información a este documento, solicitar que sea añadido algún tópico que le interese, corregir defectos o hacer sugerencias me puede contactar y sin dudas los comentarios se tendrán presente en la próxima versión de este escrito.

Dentro del código fuente

Cazabe está conformado por los siguientes 4 módulos:

Core: está pensado para ser el núcleo del sistema, donde se hace todo el trabajo duro. Contiene toda la información de la película (ancho, alto, color de fondo, etc), líneas de tiempo (la principal y la de cada símbolo), etc. Dentro de core/ la clase EventDispatcher es la encargada de “darle la cara” a la interfaz gráfica de usuario y de delegar las peticiones procedentes del módulo GUI a las clases específicas dentro de core/.

GUI: todo con lo que el usuario se relaciona directamente. No almacena datos sino que se limita a mostrar la información leída desde core/. Está conformada por los submódulos Toolbox, Timeline, PropertyEditor, GraphicsEditor, CodeEditor, Dialogs y Library. Gran parte del módulo Timeline fue tomado del proyecto UIRA y adaptado en gran medida debido a que fue escrito con otro diseño en mente; no fue pensado para solo “leer” las líneas de tiempo desde un núcleo. Planeamos reescribirlo completo en algún momento.

Cz: encargado de manipular los archivos en formato .cz. En la clase CzManip la función save (...) crea un fichero de extensión .cz con toda la información del proyecto. La función load(...) toma un fichero .cz y carga el núcleo de cazabe con la información contenida en el fichero. Estas funciones corresponden a las opciones guardar y abrir de la interfaz gráfica de usuario respectivamente.

TinyXml: software de terceros para manipular ficheros XML. Usado para facilitar la interacción con los archivos .cz.

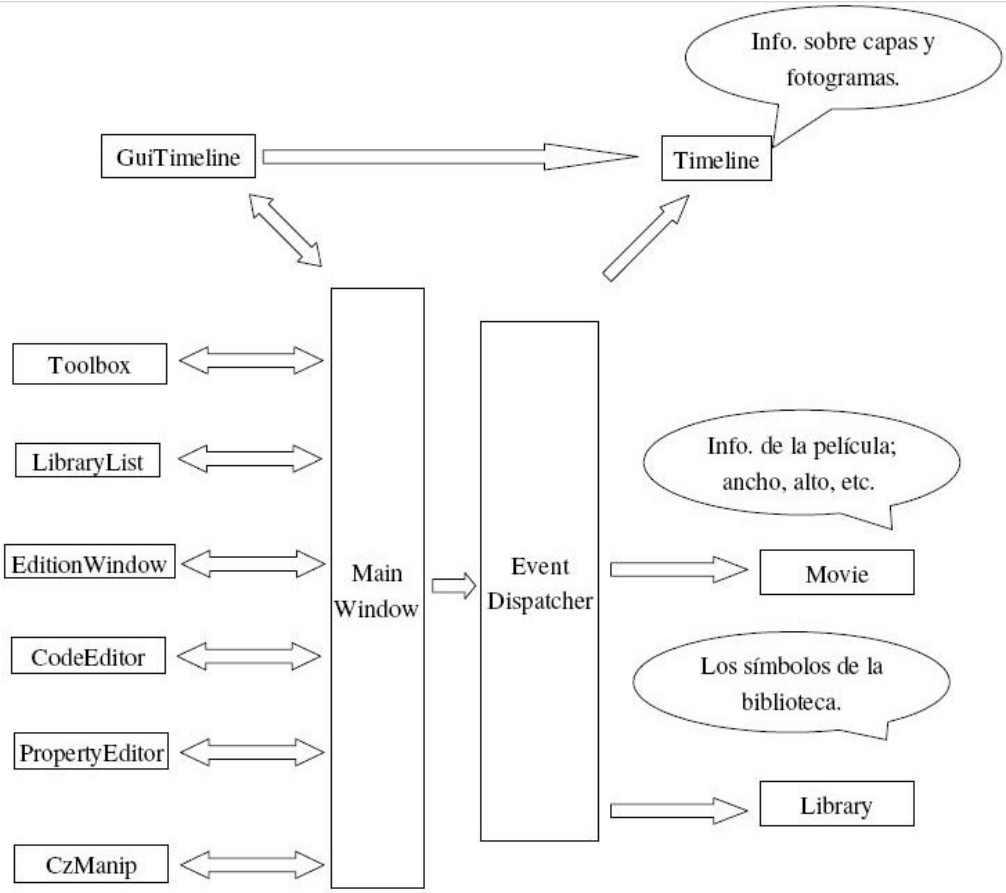

Figura 0: Vista general de la arquitectura.

Nota: Los textos encerrados en cuadrados no se refieren exactamente y en todos los casos, a clases u objetos, sino a conceptos.

La clase Scene (Core).
La clase Scene (derivada de QGraphicsScene) maneja la creación de gráficos en cazabe a través de los eventos mousePressEvent y mouseMoveEvent; Scene es el área de dibujo de cazabe. La variable miembro *mode* controla el modo de uso de la escena, por ejemplo, al hacer clic sobre el área de dibujo si el modo es InsertCircle se creará una instancia de CircleItem cuyas dimensiones cambiarán a medida que el usuario arrastre el puntero del mouse (evento mouseMoveEvent). El atributo *mode* es quien define cómo se comportará el área de dibujo ante los eventos del ratón.

En Src/Core/Shapes se encuentran todas las clases que definen las figuras. Todas las figuras tienen como ancestro común GraphicItem que a su vez hereda de QGraphicsItem (permitiendo redefinir las funciones paint(), shape() y boundingRect()) e implementa Cloneable (permitiendo redefinir clone()), de esta manera se pueden realizar las mismas operaciones sobre todos los tipos de figura pero cada cual las realiza e interpreta a su modo a través del polimorfismo (para algunos la aclaración de “polimorfismo” puede ser

obviamente una redundancia pero recuerden que el objetivo es explicar la codificación lo mejor posible). Cada figura definida en Cazabe constituye una envoltura a las clases que ofrece Qt para cada elemento gráfico (figura), por ejemplo, CircleItem tiene una agregación con QGraphicsEllipseItem logrando mantener el mismo comportamiento que define Qt para cada tipo de dibujo o modificándolo si se desea.

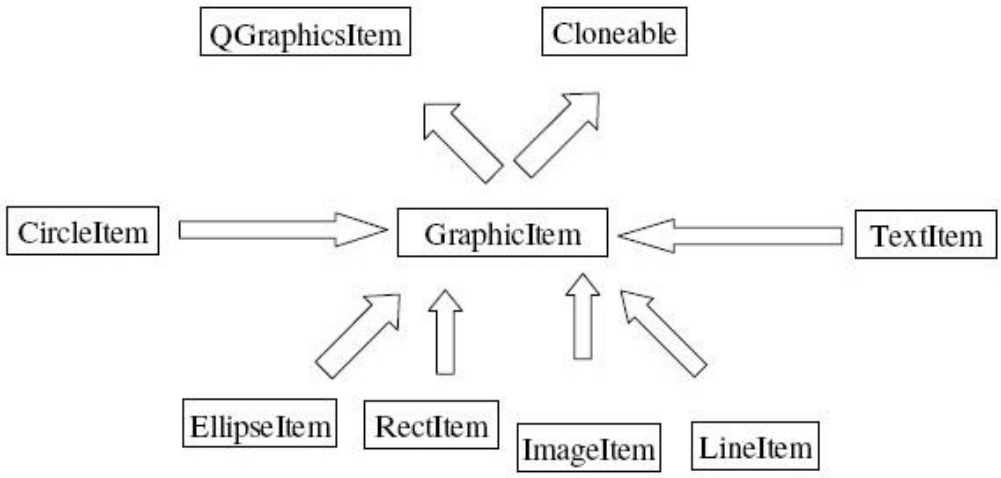

Figura 1: Estructura de clases de los elementos gráficos.

Secuencia: Seleccionar herramienta de la caja de herramientas.
Al seleccionar una herramienta de la caja en la interfaz de usuario, se cambia el valor del atributo *mode* de la clase Scene, a partir de ese instante las respuestas de Scene estarán condicionadas por el valor de esta variable.

Secuencia: Dibujar figura.
La secuencia dibujar figura se inicia al hacer clic sobre el área de dibujo de cazabe después de haber seleccionado una de las opciones de la caja de herramientas, en ese momento se crea una instancia específica de GraphicItem y se envía, a través del mecanismo signal/slot de Qt, a la capa y fotograma actual de la línea de tiempo actual del núcleo de la aplicación.

Secuencia: Probar película.
La secuencia probar película se inicia al hacer clic en Control-> Test movie.
Para visualizar una película swf, es necesario primero guardar el proyecto como un archivo cz, a continuación cazabe “le pide” a cz2swf que cree el swf a partir del fichero cz y posteriormente cazabe “le pide” a gnash que visualice la película swf. Para el manejo de cz2swf y de gnash se usa QProcess.

Extensiones o plugins

Se tiene la meta de ofrecer una arquitectura basada en plugins principalmente para extender los formatos a los que Cazabe puede exportar. Existe la posibilidad de extender por dos vías:

1- A través del formato cz. El formato .cz sería la base del resto de los formatos, si se quiere un swf se crea cz2swf y se integra a Cazabe, si se quiere un svg se programa un cz2svg, etc.
2- Exportar directamente el contenido de la línea de tiempo de Cazabe al formato deseado.

Actualmente la codificación necesaria para el soporte de extensiones está en cero.

El formato de proyectos .cz

El formato cz es, hasta el momento, el formato nativo de cazabe. Está compuesto por ficheros XML y carpetas, todo eso comprimido en un solo archivo de extensión .cz. Inicialmente se pensaba dar soporte al formato xfl de Adobe pero aún su especificación técnica no se ha publicado, así que se desarrolló un formato de proyectos propio.

cz2swf

cz2swf es una herramienta independiente (desarrollada por y para el proyecto cazabe), encargada de crear un .swf a partir de un archivo cz. Tiene una parte escrita en bash (la de descomprimir el fichero .cz con el unzip) y otra escrita en C++ (la de leer los XML y crear el swf). Parece una buena idea desarrollarla independientemente y en modo texto para no obligar a los usuarios a usar cazabe, así con cualquier editor de texto se puede crear/editar los XML y desde la línea de comandos obtener la película swf. En cazabe se usa a través de QProcess.

Bibliotecas/librerías usadas

Qt: Interfaz gráfica de usuario, manipulación de XML, procesos, etc.
Ming (libming): creación de archivos de flash (swf).

Dependencias

Para la compilación depende de Qt4 y libming 0.4.3.

Durante el proceso de ejecución se sirve de:
cz2swf: ver cz2swf en las páginas anteriores.
zip: para crear el archivo .cz comprimido.
unzip: usado para descomprimir el archivo .cz.

Conclusiones

Los elementos expuestos constituyen la base mínima para comprender las interioridades del proyecto. La presente versión del documento se sabe incompleta pero se irá enriqueciendo a través de las sugerencias y dudas de los propios lectores.

Happy Hacking.