Una aplicación de patrón de composición para realizar animaciones

Jens Hardings - Luis Mateu
Universidad de Chile
Blanco Encalada 2120, Santiago, Chile
jharding@dcc.uchile.cl

Palabras claves: animaciones, Java, doble buffering, patrón de composición.

Resumen

En la actualidad, existen diversos programas que permiten a un usuario crear animaciones de calidad en forma fácil y relativamente rápida. Sin embargo, estos programas en general presentan la desventajas como limitaciones a la flexibilidad de las animaciones o poca portabilidad.

Es por esta razón que se ha desarrollado una biblioteca que permita a los usuarios crear animaciones en forma cómoda, sin necesidad de preocuparse de los detalles comunes en las animaciones, como concurrencia, dibujado de pantalla, doble buffering, etc.

En este documento se discuten a grandes rasgos las debilidades que presentan a nuestro juicio los programas existentes para crear animaciones, y se presenta la biblioteca de animaciones en forma de un pequeño tutorial. Luego se explican los detalles del diseño y la implementación del trabajo.

1 Estado del Arte

Hoy en día hay disponibilidad de diversas herramientas que se pueden usar para crear animaciones. Con este término nos referimos a presentaciones stand-alone, que no requieren la presencia de un relator, y que pueden ser reproducidas en prácticamente cualquier computador con capacidades multimediales. Veremos algunos ejemplos de dichas aplicaciones y las razones de porqué no las consideramos ideales para nuestros requerimientos.

En primer lugar, están los ambientes para crear CD-ROMs interactivos, como el Macromedia Authorware. Estos ambientes proveen ambientes para la creación de animaciones interactivas, las

cuales han tenido éxito, sobre todo por ser la primera forma de sistemas interactivos y multimedios en masificarse. Existen diversos títulos con buena aceptación y que cumplen con objetivos como enseñar y permitir la experimentación del usuario. Sin embargo, estos sistemas requieren en general de dispositivos y recursos computacionales no estándares, lo cual limita a los creadores que deseen usar estos medios.

Por otro lado, existen herramientas para realizar presentaciones como Power Point. Estas herramientas tienen como principal objetivo el apoyar la presentación de un relator, pero también tienen la capacidad de ser vistas stand-alone, al incorporar narración y sincronización. Por lo tanto es posible utilizarlas para nuestro propósito, pero tienen la desventaja de no estar diseñadas para esos fines, y presentan desventajas claras. Por ejemplo, la manipulación de la narración y su sincronización con los elementos gráficos no está incluida directamente en la herramienta, y es necesario usar otros programas adicionalmente.

Otra restricción fundamental de los sistemas nombrados es que están en general ligados a una o dos plataformas en las cuales funcionan, y no es posible adaptarlos a nuevas arquitecturas ni sistemas operativos. Esto se suma a la restricción que muchos ambientes de desarrollo imponen sobre los elementos que es posible incluir en las animaciones, ya que tienen solamente bibliotecas predefinidas de elementos de animación.

Una posibilidad de incorporar mayor flexibilidad a las presentaciones es usar herramientas de más bajo nivel, como por ejemplo programar animaciones en Java o algún otro lenguaje de programación. Este método resuelve la limitación que presentan los otros sistemas, ya que permite ejecutar código arbitrario por parte del usuario, e incluirlo dentro de la animación. Por ejemplo, se puede incluir un programa de Chat dentro de una animación para interactuar en tiempo real con algún experto en el tema de una curso, o permitir al usuario interactuar por medio de dispositivos no tradicionales, como por ejemplo periféricos de realidad virtual.

Pero al usar un lenguaje de programación como Java, es necesario que el creador se preocupe de aspectos importantes como doble buffering, sincronización de threads, entre otros, pero que no tienen una relación directa con la creación de animaciones. Sería deseable contar con una herramienta que transmita toda la flexibilidad de un desarrollo directo en un lenguaje de programación, pero que se preocupe de administrar los elementos no directamente relacionados con la animación.

2 La biblioteca anim.operations

El resultado de este trabajo fue el desarrollo de una biblioteca de animaciones, la cual maneja todos los elementos de bajo nivel desde el punto de vista de las animaciones. Con este sistema, es fácil crear animaciones, ya que se presenta una API con la cual el autor crea elementos gráficos, y crea operaciones sobre estos elementos. Además, es posible agregar nuevas operaciones sobre elementos gráficos, programándolos en Java y agregándolos a la biblioteca de operaciones ya existentes.

Los elementos gráficos se definen como una clase llamada `Glyph`. Estos elementos pueden ser

polígonos, imágenes, texto, y cualquier otro elemento que se defina posteriormente. Las animaciones se construyen en base a operaciones sobre estos objetos, como por ejemplo la creación de un Glyph, movimiento o cambio de color.

La biblioteca puede ser encontrada en la siguiente URL:
`http://www.dcc.uchile.cl/~jharding/anim/`. En esa dirección se encuentra un archivo en formato JAR, además de las instrucciones de uso. También es posible accesar la publicación [2], en la cual se detallan los temas relacionados.

2.1 Creación de una animación

Una animación es una lista de operaciones sobre Glyphs. Las animaciones que están implementadas actualmente, son `Sequential` y `Parallel`. La primera agrupa operaciones que se ejecutan secuencialmente, es decir, la siguiente operación se ejecuta cuando ha terminado de ejecutar la anterior. Por lo mismo, la duración de esta operación será la suma de las duraciones de las operaciones que agrupa. En cambio, las animaciones agrupadas dentro de un objeto de clase `Parallel` se ejecutan todas simultáneamente. Así, la duración será igual a la máxima duración de las operaciones que se agrupan.

Si algún autor de animaciones quisiera incluir una nueva clase que por ejemplo ejecutara sólo alguna de las operaciones agrupadas, basado en interacciones con el usuario, podría implementarla e incluirla como una nueva animación.

Una propiedad importante de las animaciones es que además de ser animaciones, son también operaciones. Con esto es posible crear una biblioteca personal de animaciones predefinidas, las cuales pueden ser usadas dentro de nuevas animaciones como si fueran operaciones simples. Así, una animación se puede ver como un árbol en el cual los nodos son animaciones, y las hojas son operaciones simples.

Toda la animación se ejecuta dentro de un objeto de clase `Theater`, que es la encargada de proveer el `Canvas` donde se pintarán los elementos gráficos.

Para crear una animación simple, se usaría el siguiente código:

```
Theater teatro = new Theater();
Animation hola_mundo = new Sequential();
```

Luego es posible agregar un elemento gráfico, con la siguiente línea:

```
GPoint inicial = new GPoint(10,10);
Glyph g = new GText("Hola, Mundo", teatro, inicial);
Creation texto = new Creation(g);
hola_mundo.add(texto);
```

Lo siguiente que se podría hacer es esperar dos segundos y luego mover el texto a otro punto. Ese movimiento se define con una duración de 6 segundos.

```
hola_mundo.add(new Pause(texto,2));
GPoint final = new GPoint(50,50);
hola_mundo.add(new Movement(texto, 6, inicial, final));
```

Esta animación puede ser usada como cualquier otra operación. En particular, puede ser usada dentro de otra animación, o bien se puede guardar en un Stream, ya sea una conexión de red o un archivo, entre otros, de la siguiente manera:

```
ObjectOutputStream out = ...;
out.writeObject(hola_mundo);
```

Una vez hecho esto, se puede reusar una animación guardada para reproducirla, o para utilizarla dentro de otra animación.

A continuación se agrega el código que genera una animación un poco más compleja:

```
GColor rojo = new GColor(255,0,0);
Theater teatro = null;
Animation anim = new Sequential();
Creation flecha = new Creation(
                        new GArrow(100., -130., rojo),
                        teatro, 90., 170.);
anim.add(flecha);
Creation texto = new Creation(new GText("Texto a seguir"),
                        teatro, 150., 30.);
anim.add(texto);

Animation par = new Parallel();
par.add(new Movement(flecha, 3, 90., 170., 140., 280.));
par.add(new Movement(texto, 3, 150., 30., 200., 140.));
anim.add(par);

anim.add(new Pause(teatro, 2));

par = new Parallel();
par.add(new Movement(texto, 3, 200., 140., 400., 140.));
par.add(new GrowingArrow(flecha, 3, 100., -130., 300., -130.));

anim.add(par);
```

El código primero crea dos glyphs, una flecha y un texto. Luego, se mueven ambos elementos del punto inicial a otro final, en un tiempo de 3 segundos. Esto último se realiza usando una animación de tipo Parallel. Acto seguido, se introduce una pausa de 2 segundos, y luego se agrega una nueva animación paralela. Esta última animación mueve el texto, y a su vez hace crecer la flecha, para que "siga" al texto. Esta animación se puede visualizar como el arbol de la figura 1.

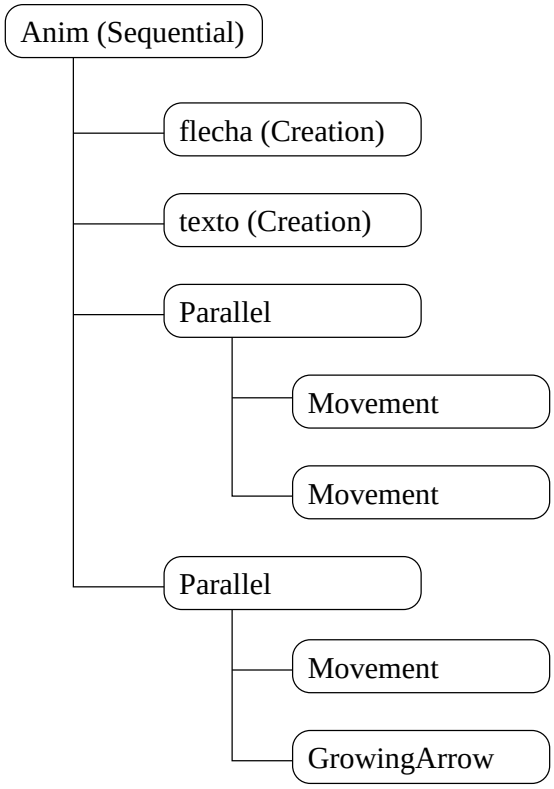

Figura 1: Arbol generado

2.2 Reproducción de una animación

Una vez que se cuenta con una animación en un archivo, es posible reproducirla usando un pequeño programa en Java que abre una ventana, lee el archivo y despliega la animación. El usuario controla el despliegue de la animación usando botones que presentan una analogía a un videograbador. Es posible reproducir toda la animación usando el botón `play`, y en cualquier momento se puede detener presionando `pause`.

También existe la posibilidad de navegar a través de la animación avanzando solamente una operación, presionando un botón playOne(). Cabe hacer notar que el avanzar una sola operación

puede significar ejecutar una animación completa. Un resumen de las funcionalidades que provee esta interfaz es el siguiente:

- `play`: reproduce la ejecució de la operación.
- `pause`: interrumpe la ejecución, dejando la operación en un estado intermedio. Este estado puede ser continuado para llegar al estado final presionando los botones `play` o `advance`.
- `stop`: interrumpe la ejecución, y vuelve la operación a su estado inicial.
- `advance`: avanza la operación. Tiene la misma funcionalidad que `play`, pero sin realizar la animación, y termina en el menor tiempo posible.
- `back`: retrocede la operación al estado inicial.
- `playOne`: reproduce solamente una de las operaciones de una animación. Para el caso de operaciones básicas, tiene el mismo efecto que `play`.
- `advanceOne`: avanza solamente una operación de una animación, sin mostrar los estados intermedios (es decir, en el menor tiempo posible). Para el caso de operaciones básicas, tiene el mismo efecto que `advance`.
- `backOne`: retrocede solamente una operación de una animación, sin mostrar los estados intermedios (es decir, en el menor tiempo posible). Para el caso de operaciones básicas, tiene el mismo efecto que `back`.

3 Diseño

El objetivo del diseño es proveer una API que permita mantener la flexibilidad y el potencial que provee un lenguaje de programación como Java, pero a la vez liberar al programador de tareas tediosas y poco relacionadas con las animaciones que pretende mostrar. Para ello, se usó el patrón de diseño denominado *Composite* en el libro "Design Patterns" [1]. La idea básica del patrón de diseño es crear un modelo en el cual existen clases de tipo *componente* y clases que agrupan *componentes*, llamadas *compositoras (Composite)*. Estas clases pueden compartir una interfaz común, representada por una superclase. Así, no es necesario diferenciar en general entre clases *componentes* y clases *compositoras*. Especificamente, en la biblioteca que se desarrolló, las clases compositoras son denominadas *animation* y las componentes *operation*.

En el diagrama de la figura 2 se observa que todas las clases definidas son subclases de `Operation`, incluso las clases que componen operaciones, que son subclases de `Animation`. Esto permite que cualquier animación pueda ser usada como si fuera una simple operación. Las clases `Creation`, `Movement` y `GrowingArrow` son ejemplos de operaciones simples.

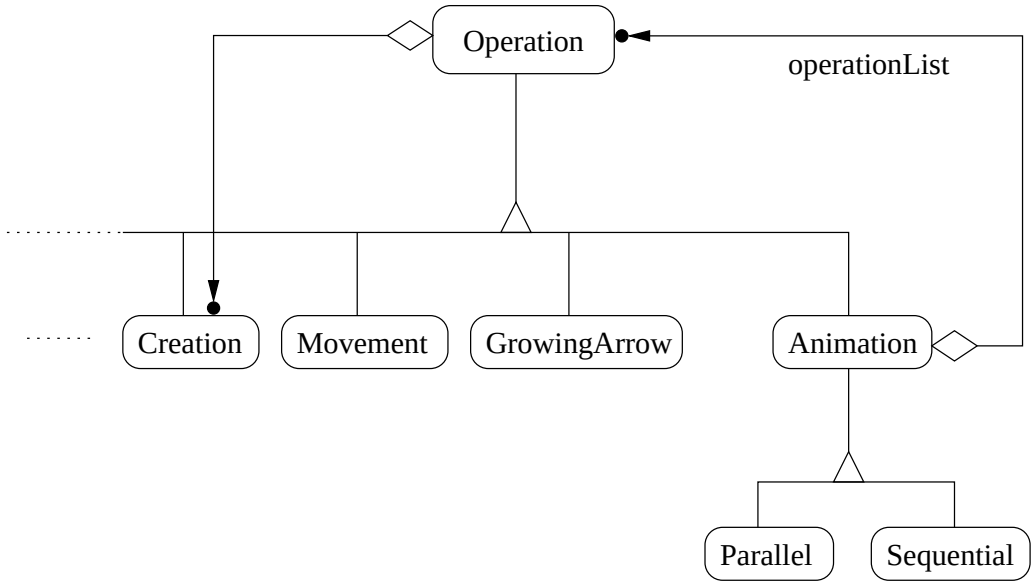

Figura 2: Diagrama de Clases de anim.operations

Un aspecto importante en el diseño es que finalmente la herramienta estará orientada a una interfaz gráfica, la cual será usada por el creador del curso envasado. Es por ello que se ha definido que las operaciones deben ser interruptibles y reversibles. Estos requerimientos son necesarios para permitir una interacción en forma gráfica con el usuario, en la cual se puede presentar cualquier estado de la presentación en un momento dado, modificarlo y continuar con la reproducción.

Para facilitar la inclusión de nuevas operaciones a la biblioteca de operaciones con la que cuenta el usuario, se trató de exigir lo menos posible a las operaciones.

4 Implementación

La implementación de la biblioteca trata de incluir todos los aspectos necesarios para el correcto funcionamiento de las animaciones, pero sin restringir las funcionalidades que pueden tener las mismas. A continuación se revisan aquellos temas que son lo suficientemente genéricos para ser manejados de igual manera para todas las animaciones que se pretenden crear usando la biblioteca desarrollada.

En una animación, como se define en la biblioteca, participan diversas operaciones, las cuales pueden necesitar ejecutarse en paralelo. Por ello, cada operación se ejecuta en su propio thread y es necesario proteger diferentes threads de accesar estructuras de datos compartidas. Para lograr esto, se ha implementado un sistema que ejecuta los threads en forma *non-preemptive*. Con esto se garantiza que siempre haya un solo thread del grupo ejecutándose, y no es posible interrumpirlo

para seguir la ejecución de otros threads.

Al crear una animación, automáticamente se define un tiempo de simulación. Este tiempo establece en qué momentos se realiza alguna acción u ocurre un evento en la animación. Al reproducir la animación, idealmente debiera coincidir siempre ese tiempo de simulación con el tiempo real, pero basta con que estos tiempos coincidan en los instantes en que se dibuja la pantalla. Con este método, se redibuja la pantalla una cantidad razonable de veces por segundo como para dar la impresión de un movimiento continuo. En general, el tiempo de simulación se puede avanzar más rápido que el tiempo real, con lo cual se introducen pausas en la animación. En caso de que el tiempo de simulación no puede ser avanzado lo suficientemente rápido como el tiempo real, se debe elegir entre atrasar el tiempo de simulación o ignorar algunos dibujos de la pantalla para mantenerlo sincronizado. Por lo anterior, y dado que las operaciones funcionan en threads independientes, es necesario contar con un mecanismo de sincronización para que las operaciones tengan todas la misma noción de tiempo de simulación. Para lograrlo, existe un thread encargado de administrar una agenda de eventos, la cual determina qué thread necesita ser despertado para realizar su parte dentro de la animación. Toda la sincronización entre la agenda de eventos y las operaciones, se maneja mediante mensajes.

La biblioteca implementa el mecanismo de *doble buffering* para el dibujado de la pantalla. Este consiste en realizar el dibujado en una estructura gráfica que no se presenta en la pantalla, y copiar la imagen a pantalla una vez que el dibujado se haya completado. Usando este mecanismo, se evita que se vean en la pantalla los progresos del dibujado, lo cual generaría una sensación de "flickering".

5 Conclusiones

Normalmente se supone que todo programa que realice una animación debe ser intrínsecamente complicado. En este trabajo se muestra que esto no es cierto. En el trabajo se desarrolló una biblioteca que abstrae los elementos fundamentales de una animación en la `Operation`, y que oculta del programador todos los detalles de implementación de características más avanzadas como flickering, la concurrencia, dibujo, etc. Con la biblioteca se pueden lograr animaciones de buena calidad sin demasiado trabajo, y teniendo además toda la flexibilidad que ofrece un lenguaje de programación.

Referencias

[1] *Design Patterns, Elements of Reusable Object-Oriented Software*, Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Addison-Wesley, 1999.

[2] *Prototipo de un ambiente de desarrollo para animaciones en Java para cursos envasados*, Jens Hardings Perl

[3] *http://www.macromedia.com/*, Macromedia Director

[4] *http://www.macromedia.com/*, Macromedia Authorware