Diseño de Locker Room

Carlos Martín Villanova, Tonghong Li,
Claudio Soriente, Ricardo Jiménez Peris and Marta Patiño Martínez

El diseño de *LockerRoom* está basado en patrones de diseño, que son una solución re-usable para un problema común de diseño. Los patrones de diseño no son un algoritmo que se pueda transformar directamente del código, es una descripción o una plantilla que resuelve un problema que se puede usar en muchas situaciones diferentes. En general los patrones de diseño de programación orientada a objetos muestran las interacciones entre clases u objetos sin especificar la aplicación de estos.

Los patrones de diseño aceleran el desarrollo al proveer paradigmas comprobados. El diseño efectivo de software requiere que se consideren aspectos que pueden no aparecer hasta la fase de implementación. El re-usar los patrones de diseño ayuda a prevenir los sutiles incidentes que se pueden convertir después en problemas. También facilita la lectura para los profesionales que están familiarizados con ellos. Al programar con patrones de diseño hay que tener en cuenta que para conseguir flexibilidad introducen más niveles de abstracción, lo cual puede causar que los diseños sean más complicados y perjudicar al rendimiento de la aplicación.

Por definición un patrón tiene que programarse cada vez que se usa en una aplicación, esto según algunos autores es un paso atrás en la reusabilidad del código que proveen los componentes.

Otra característica de los patrones de diseño es que proveen soluciones generales documentadas en un formato que no requiere información específica a un problema particular.

Los patrones están compuestos de varias secciones, siendo de particular interés las secciones de estructura, participantes y colaboración. Estas secciones describen una micro-arquitectura que los desarrolladores copian y adaptan a su diseños para solucionar los problemas resueltos por los patrones de diseño. Esta micro-arquitectura es un conjunto de partes del programa (clases, métodos,...) y sus relaciones.

Los desarrolladores usan los patrones de diseño al introducir estas micro-arquitecturas en sus proyectos. Esto significa que los diseños tendrán una estructura y organización similar al patrón utilizado. Además los patrones permiten a los desarrolladores comunicarse utilizando nombres conocidos y entendidos por todos para la interacción de software.

Los patrones de diseño se agruparon inicialmente en las categorías patrones creacionales, patrones estructurales y patrones de comportamiento, a las que se añadió más tarde patrones de concurrencia. Los patrones se describen usando los conceptos de delegación, agregación y consultación.

Los patrones creacionales son los que están relacionados con la inicialización de las clases. Se pueden dividir a su vez en patrones de creación de clases y de creación de objetos. La diferencia es que mientas que los patrones de creación de clases usan herencia en el proceso de inicialización, los de creación de objetos usan delegación para hacer su función.

Los patrones estructurales son los que facilitan el diseño al identificar una manera simple de modelizar las relaciones entre entidades.

Los patrones de comportamiento son los que identifican patrones comunes de comunicación entre objetos e implementan estos patrones. Al hacer esto los patrones aumentan la flexibilidad para establecer estas comunicaciones.

Los patrones concurrentes son los que se ocupan del paradigma de programación concurrente.

En un diseño se utilizan patrones de distintas categorías ya que estás solucionan distintos tipos de problemas de diseño.

Patrones Utilizados

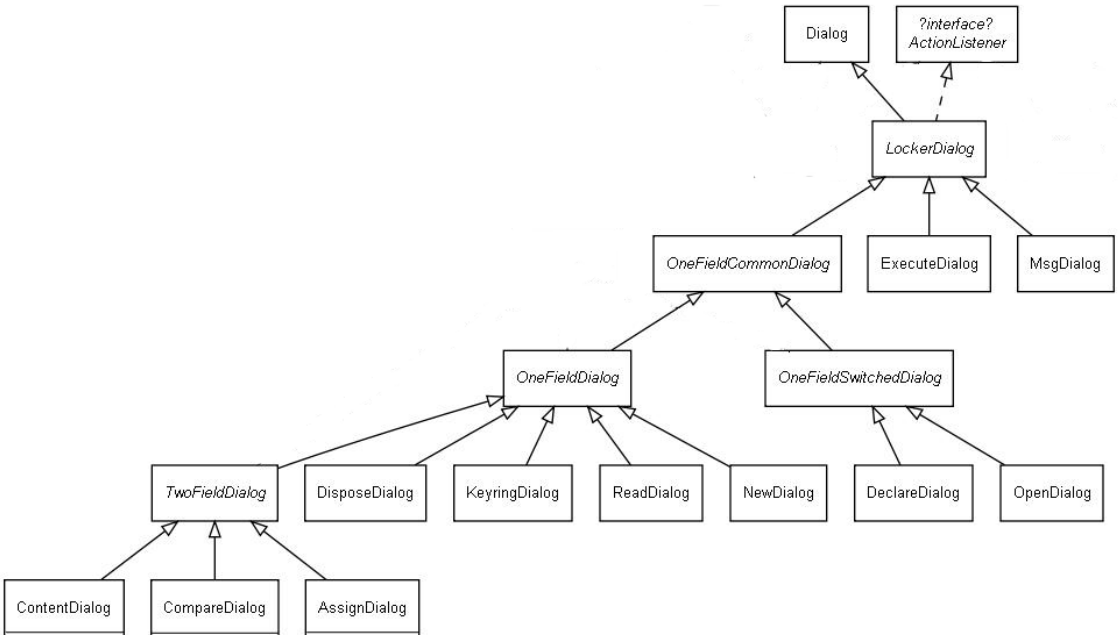
LockerRoom utiliza la librería *Xtangoanimator* para la animación de los objetos. Dado que esta librería es genérica se ha modificado para que pudiera representar los objetos del *LockerRoom*, tanto *LockerRoom* como *Xtangoanimator* utilizan patrones para solucionar varios problemas de diseño.

LockerRoom está escrito completamente en java lo que implica que se utilizan los patrones intrínsecos en java. Estos patrones están implementados en las librerías y utilidades. Por ejemplo al recorrer listas hay un patrón *Iterator* implícitamente.

El diseño de *LockerRoom* ha tenido que solucionar un considerable número de problemas de diseño de varios tipos, para explicar los patrones utilizados en cada caso en preferible utilizar las categorias de patrones descritas por Gamma, Erich; Richard Helm, Ralph Johnson, and John Vlissides en Design Patterns: Elements of Reusable Object-Oriented Software y así seguir una estructura conocida.

- **Patrones Creacionales**

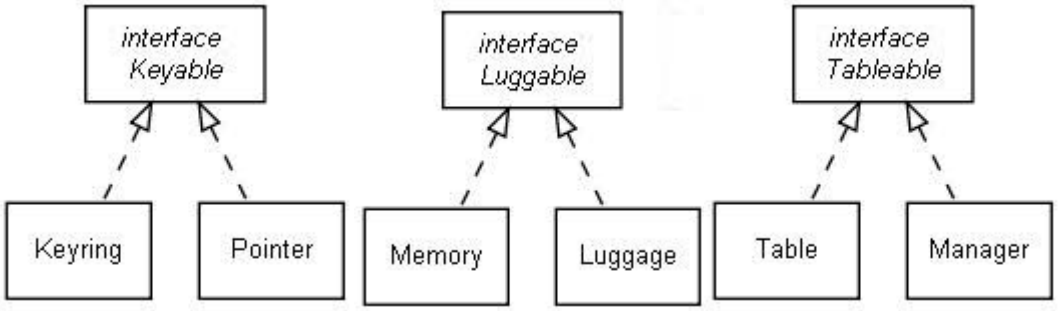
Los patrones usados de esta categoría se utilizan para solucionar problemas al definir las clases. El primero es que era necesario tener un numero de diálogos distintos para cada operación del *LockerRoom*, estos diálogos tienen muchas propiedades en común, pero se tienen que ejecutar distintas operaciones, la solución para esto es el patrón *Factory*, que permite definir un interfaz para crear el objeto, pero delega en la subclase la decisión de que clase se instancia finalmente. La implementación de este patrón es en *LockerDialog* y sus subclases.



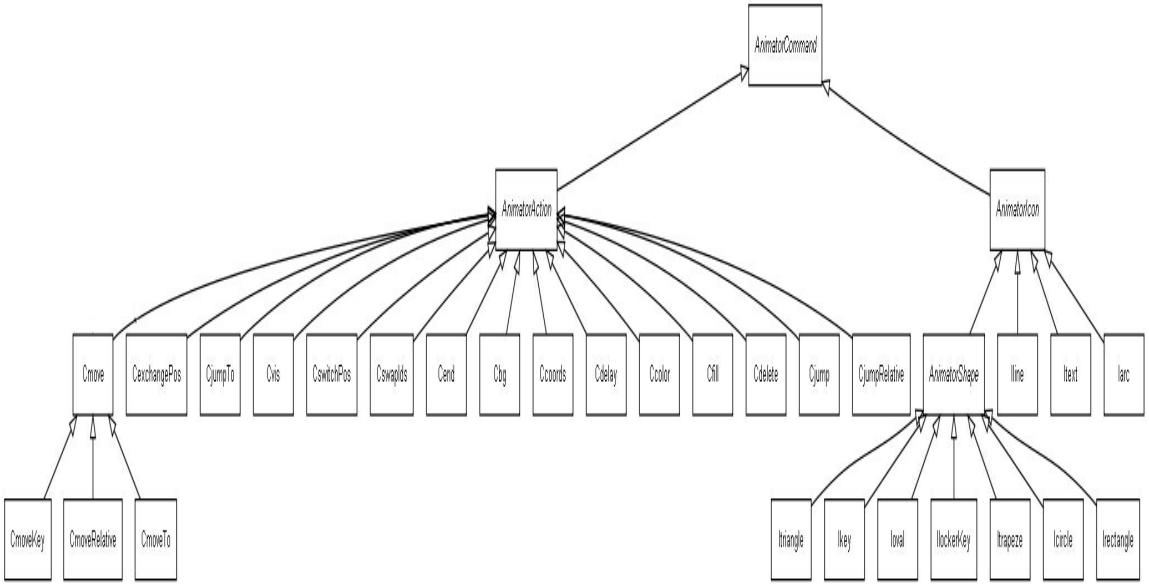
El segundo problema era que al tener dos representaciones, metáfora y programación, era necesario tener dos clases que tienen las mismas funciones pero mostrando distintos gráficos. Para hacer esto sería necesario repetir el código tantas veces como representaciones fueran

posibles. La solución a esto es el patrón *AbstractFactory* que permite definir los métodos sin tener que decidir la implementación hasta el momento de ejecución.

Este patrón es de uso general en el diseño por la naturaleza del programa, todas las clases que representan un objeto de programación (*Manager, Memory, Pointer,...*) y las que representan un objeto de la metáfora (*Table, Luggage, Keyring,...*) están basados en este patrón.



La librería *LockerAnimator* también está basada en este patrón, el problema a solucionar en este caso es que la librería ofrece un número de objetos y operaciones que tienen un interfaz común para facilitar el uso.

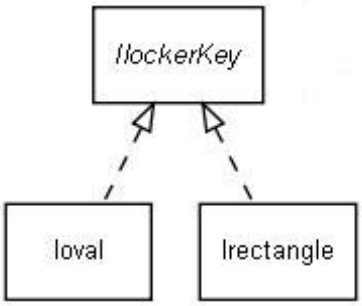


- **Patrones Estructurales**

Los patrones estructurales se utilizan para resolver problemas de relación de entidades, en este diseño el primer problema era tener un interfaz simple para usar todas las funcionalidades de la aplicación. Para resolver este problema se ha usado el patrón *Facade* que es la clase *Locker*. Esta clase tiene toda la información global que necesitan los distintos componentes para operar. Este patrón también se usa en la librería *Xtangoanimator* con el mismo fin.

Otro problema que entra dentro de la categoría de estructural es como organizar los distintos diálogos. Estos objetos comparten un número de propiedades pero también tienen diferencias, esto implicaría tener un número muy grande de objetos si cada una de las posibilidades fuera un objeto. El patrón *Decorator* encaja perfectamente con esta situación y está implementado en todas las clases de diálogo.

El último problema solucionado con un patrón de esta categoría es que para representar una llave en la metáfora eran necesarios manejar muchos objetos a la vez, circulo, rectángulo, línea,... Esto complicaba mucho la programación. La solución fue utilizar el patrón *Composite* que sirve para unir varios objetos en uno, esto lo implementa la clase *iLockerKey*.



- **Patrones de Comportamiento**

Este tipo de patrones son utilizados a la hora de modelizar el comportamiento de los objetos. Como un principio general el patrón *Command* se ha usado para resolver varios problemas. En este patrón, un objeto se usa para representar y encapsular toda la información para llamar a un método más tarde. Esta información incluye el nombre del método, el objeto al que pertenece y los valores de los parámetros. En la librería de java Swing una acción es un *Command*. Aparte de la habilidad para ejecutar el comando una acción puede tener asociado un icono, texto, información, etc. Un botón o elemento de un menú son objetos de este tipo, lo que implica que este patrón se utiliza implícitamente en todos los botones de la aplicación.

Otro uso del patrón *Command* es que sí todas las acciones del usuario están representadas por objetos de este tipo, un programa puede grabar la secuencia de acciones, al mantener la lista de los objetos "comand" tal y como se ejecutan. Entonces las acciones se pueden repetir al ejecutar otra vez los objetos "command" en secuencia. Éste es el método utilizado en la clases *InputFrame* para grabar y luego ejecutar los comandos automáticamente. La clase *InputFrame* también usa el patrón *Interpreter* para solucionar el problema de evaluar las sentencias de comandos y traducirlas a java.

En esta categoría está también el patrón *Iterator* el cual se usa para acceder a los elementos de un objeto complejo sin tener que conocer la representación interior. El *Iterator* encapsula la estructura interna de como la iteración ocurre. Esto es como la clase ListDialog implementa el recorrido de la cadena de punteros.

- **Patrones Concurrentes**

El diseño de *LockerRoom* es concurrente en el sentido de que se ejecuta un nuevo hilo de ejecución por cada animación que se ejecuta, y de que todas estas animaciones siguen el patrón *Active*, que separa la ejecución del método de la invocación que reside en su propio hilo de ejecución. El objetivo es introducir concurrencia al usar un método de invocación asíncrono y un planificador para manejar las peticiones.