Capítulo 2

|  |  |  |
| --- | --- | --- |
| Jonathan Zinzan Salisbury Vega  *Universitat de les Illes Balears* Palma, España jonathan.salisbury1@estudiant.uib.cat | Joan Sansó Pericás  *Universitat de les Illes Balears* Palma, España joan.sanso4@estudiant.uib.cat | Joan Vilella Candia  *Universitat de les Illes Balears* Palma, España joan.vilella1@estudiant.uib.cat |

Conceptos: Recursividad backtracking, recorregut euleria (casella unica)

*El coste asintótico de los algoritmos ha sido uno de los principales focos de estudio (y de problemas) a lo largo de la historia de la informática. Las soluciones más sencillas suelen tener costes asintóticos no deseables dependiendo del problema a resolver. Además, encontrar soluciones óptimas suele traer verdaderos quebraderos de cabeza a los distintos programadores. En este proyecto se ha implementado una aplicación capaz de visualizar los distintos tiempos que tienen estos algoritmos. Para ello se ha utilizado el Modelo Vista Controlador. Utilizar esta arquitectura permite un mayor control de errores del programa y facilidad de reutilización como de escalado. Todo esto a coste de una mayor complejidad de código. Con esta práctica se busca poner en práctica la arquitectura MVC y ofrecer una herramienta al programador para poder discernir de una manera gráfica con la que estudiar la viabilidad de los distintos algoritmos.*

Keywords—MVC, rendimiento, coste asintótico, algoritmo.

# Introducción

Una de las máximas que ha de tener un programador a la hora de diseñar sus algoritmos, es el coste asintótico de los mismos. Muchas veces, por un simple descuido o por desconocimiento sobre la materia, se acaban implementando soluciones muy alejadas de lo que podríamos considerar como aceptable.

La herramienta que se ha desarrollado permite visualizar de una manera más fácil el coste asintótico de los distintos órdenes de complejidad. Esto permite al usuario hacerse una idea de la viabilidad de sus soluciones, no necesariamente para que sean las óptimas, pero sí para que se puedan realizar en un tiempo aceptable.

El segundo objetivo de la práctica ha sido la puesta en práctica del MVC (Modelo Vista Controlador) visto en clase. La estructura del documento será la siguiente, en primer lugar, se describe la implementación del modelo MVC y después se comentan los resultados de los distintos costes asintóticos.

# Modelo vista controlador

Antes de explicar el modelo vista controlador, es importante entender qué es un patrón de diseño (o arquitectura).

Un patrón de diseño es un conjunto de técnicas que se estandarizan para resolver un conjunto de problemas comunes en el desarrollo de software. No toda técnica puede ser considerada un patrón de diseño, por lo que debe cumplir las siguientes características:

* Efectividad ante problemas similares.
* Reutilizable

Por lo que un patrón podría ser considerado una plantilla en la que se unifican una serie de buenas prácticas. El Modelo Vista Controlador no es más que un patrón de entre todos los que hay. Pero recoge una serie de particularidades especialmente útiles para el diseño de esta práctica.

El consenso general ha dividido los patrones de diseño en tres grandes grupos.

* Creacionales
* Estructurales
* Comportamiento

Los creacionales están enfocados en la producción de objetos bajo una serie de criterios. Los estructurales se encargan en la organización de clases y objetos para brindar una solución y finalmente, los de comportamiento identifican similitudes entre la interacción de clases y objetos. Además, existe una distinción para los patrones con soluciones basadas en concurrencia.

Basándonos en las definiciones antes dadas, es difícil colocar al MVC en uno de los tres grupos. Esto es porque este patrón es de “dominio específico”, por lo que está muy orientado a las interfaces de usuario. Dentro de este grupo también se pueden encontrar soluciones específicas de: la seguridad informática, modelos de negocios, entre otros.

## Historia y definición original

El MVC lo crea Trygve Reenskaug, ingeniero informático y profesor emérito en la Universidad de Oslo, durante el desarrollo del lenguaje Smalltalk-79, a finales de los 70. Después de varios diseños, opta por el actualmente conocido MVC.

En propias palabras de Trygve Reenskaug: *“El propósito esencial de MVC es cerrar la brecha entre el modelo mental del usuario humano y el modelo digital que existe en la computadora. La solución MVC ideal respalda la ilusión del usuario de ver y manipular la información del dominio directamente. La estructura es útil si el usuario necesita ver el mismo elemento del modelo simultáneamente en diferentes contextos y/o desde diferentes puntos de vista. La siguiente figura ilustra la idea.”*

Diagram

Description automatically generated

Como se puede observar en la definición original, el objetivo primordial de este modelo es facilitar el entendimiento al usuario final (modelo mental), facilitando así la conexión con el modelo. Es importante tener esto en cuenta porque muchas veces se desvirtúa la intención de este patrón, pensando que es una composición elegante de tres instancias.

## Definición actual y ventajas

La definición más aceptada a día de hoy es la propuesta por el libro Design Patterns: Elements os Reusable Object-Oriented Software (1994):

“*MVC consta de tres tipos de objetos. El modelo es el objeto de la aplicación, la vista es su presentación en pantalla y el controlador define la forma en que la interfaz de usuario reacciona a la entrada del usuario. Antes de MVC, los diseños de interfaz de usuario tendían a agrupar estos objetos. MVC los desacopla para aumentar la flexibilidad y la reutilización.*”

Como podemos observar, en esta definición ya se introduce el concepto de objeto, el cual es primordial en la programación actual. Además, ofrece una aproximación más actualizada de la función de cada una de las partes del patrón MVC.

Otra de las características más importantes, que no se mencionan en las anteriores definiciones, es el establecimiento de un protocolo suscripción/notificación entre los tres objetos. Es decir, cada vez que los datos del modelo cambien, el modelo deberá notificar a todas las vistas que dependen de él. En respuesta, las vistas deberán de actualizarse. Esto permite tener múltiples vistas adjuntadas a un modelo, en el que se podrán hacer diferentes presentaciones de la información.

Recapitulando todas las ventajas clave del patrón:

* Capacidad de adjuntar múltiples vistas para un modelo, permitiendo así distintas representaciones.
* Facilidad a la hora de cambiar la forma en que una vista responde a la entrada del usuario sin cambiar su presentación visual.
* Reutilización de código.

Ahora bien, este patrón añade una complejidad a la hora de programar. Ya que se debe realizar una separación en tres objetos a la hora de plantear una solución al problema. Si no se necesita realizar este desacoplamiento en tres objetos, se está optando por una complejidad innecesaria.

## Contexto de estudio

# Bibliografía

|  |  |
| --- | --- |
| [1] | Oracle, «UISwing Concurrency - Swing Worker, » [En línea]. Available: https://docs.oracle.com/javase/tutorial/uiswing/concurrency/worker.html. |
| [2] | Wikipedia, «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Divide-and-conquer\_algorithm. |
| [3] | «Oracle Java Documentation - SwingWorker,» [En línea]. Available: https://docs.oracle.com/javase/7/docs/api/javax/swing/SwingWorker.html. |
| [4] | J. M. Alarcón, «Rendimiento de Algoritmos y Big O - Campus MVP,» [En línea]. Available: https://www.campusmvp.es/recursos/post/Rendimiento-de-algoritmos-y-notacion-Big-O.aspx. |
| [5] | P. M. H. Thomas V. Perneger, «Writing a research article: advice to beginners - Oxford Academic,» [En línea]. Available: https://academic.oup.com/intqhc/article/16/3/191/1814554. |