PROCEDIMIENTO DE DISEÑO DE UN SISTEMA DE GESTIÓN Y CONTROL DE FLOTAS HETEROGÉNEAS

Miguel Rodríguez Caudevilla, Javier Torres López-Sepúlveda,
José Manuel Pardo Martín, Francisco Javier Ortega González, César Benavente Peces
miguelrc@alumnos.upm.es, jtorres@gpss.euitt.upm.es, jmpardo@diac.upm.es, fjortega@diac.upm.es, cbpeces@ics.upm.es
G.I.R.A. - E.U.I.T.Telecomunicación - U.P.M. - Ctra. Valencia km.7 28031.

***Resumen*—The purpose of this study is to set up the basis needed to develop real applications designed for the control and management of heterogeneous fleets. This paper tries to present some programming patterns which will lead to the design and development of good quality applications. The premise for this purpose should be the design of scalable, platform independent, modular and user friendly programs which require very small maintenance and if possible to give the ability to non-programmer users the ability to extend the application beyond their original design. To achieve it, a web design with the use of content management system following a model-view-controller architectural pattern has been purposed.**

I. Introducción

A lo largo de los últimos años ha quedado patente como la industria de las telecomunicaciones se ha ido introduciendo en la gestión y control de flotas, gracias a los avances en las comunicaciones mediante satélite, comunicaciones inalámbricas, comunicaciones móviles, etc. Estas tecnologías se han instalado en los vehículos de las flotas terrestres, marítimas y aéreas con el fin de poder aumentar la eficacia de los servicios prestados y poder llevar un control de cada uno de los vehículos de la flota. Sea cual sea el servicio prestado, son múltiples los beneficios que otorgan los sistemas de comunicaciones: control del tráfico rodado para servicios de última milla, control del flujo de embarcaciones en los puertos marítimos, adquisición de datos climatológicos, etc.

Atendiendo a esta tendencia, este trabajo trata de sentar las bases a la hora de diseñar aplicaciones para el control y gestión de flotas heterogéneas. Las aplicaciones que deben dar este servicio deben ser robustas, escalables, independientes de plataforma, modulares y con una interfaz clara y amigable para su uso por un equipo sin conocimientos técnicos sobre el funcionamiento interno del servicio. Para ello, se ha planteado hacer uso de los gestores de contenidos web que hacen uso de patrones de diseño basándose en la separación modelo-vista-controlador.

II. El procedimiento de diseño Modelo-Vista-Controlador

Cuando se diseña una aplicación es recomendable tener en cuenta las conductas de programación que se van a seguir. Estas conductas pueden desembocar en programas más fáciles de mantener gracias a una estructura y jerarquía del proyecto más clara, más escalables permitiendo abarcar un conjunto de servicios más heterogéneos y mas modulares lo que puede ayudar a segmentar y asignar distintas tareas a un grupo diferente de programadores. En muchas ocasiones, estas conductas vienen impuestas por el tipo de aplicación que se quiere desarrollar o por las herramientas utilizadas para desarrollarla, pero en cualquier caso es un tema a tratar antes de embarcarse en la programación de cualquier aplicación.

En la figura 1 se muestra la estructura tradicional de programación de una aplicación que requiere la intervención de un usuario. Este procedimiento puede presentar varias desventajas cuando se desarrolla una aplicación. En un servicio enfocado a la gestión de flotas heterogéneas, se hace uso de datos muy diversos en función de la flota que se quiere gestionar pero la lógica del gestor puede que ser la misma para las diferentes flotas. Si se sigue este procedimiento de diseño, debido a la heterogeneidad de los datos, se puede acabar con un programa difícil de mantener y de escalar. Por este motivo, se presenta la posibilidad de seguir un patrón de diseño basado en la separación de modelo-vista-controlador (MVC).

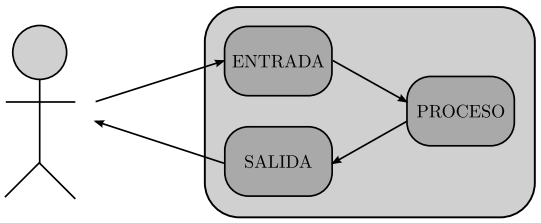

Fig. 1. Procedimiento tradicional de diseño de aplicaciones.

Las conductas de diseño de programas basadas en MVC consisten en separar los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos (ver figura 2). El estilo fue descrito por primera vez en 1979 por Trygve Reenskaug, entonces trabajando en Smalltalk en laboratorios de investigación de Xerox [1]. La implementación original está descrita en *"Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)"* [2]. Su uso se ha extendido más en el desarrollo de aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página, el modelo es el sistema de gestión de base de datos y la lógica de negocio y el controlador es el responsable de recibir los eventos de entrada desde la vista.

El diseño MVC asigna a los objetos de una aplicación cada uno de los tres roles posibles: modelo, vista o controlador. El patrón MVC no sólo define el papel que juegan los objetos en

la aplicación, también define la forma en la que los objetos se comunican entre sí. Cada uno de los tres tipos de objetos se separa de los demás por fronteras abstractas y se comunica con el resto de objetos a través de interfaces bien definidas.

El procedimiento MVC es fundamental para un buen diseño. Los beneficios de la adopción de este modelo son numerosos. Muchos objetos en este tipo de aplicaciones tienden a ser más reutilizables, y sus interfaces tienden a estar mejor definidas. Las aplicaciones que tiene un diseño MVC son también más extensibles a otras aplicaciones de distinta índole.

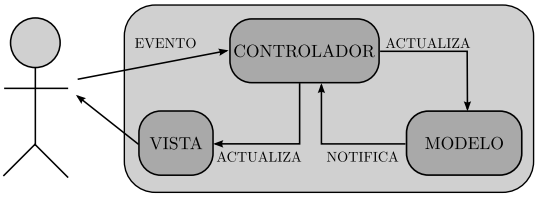

Fig. 2. Procedimiento de diseño siguiendo una arquitectura MVC.

A. El modelo

1) Implementación: Estos objetos encapsulan los datos específicos de una aplicación y definen la lógica y el cálculo que los manipula y procesa. Un modelo puede tener relaciones con otros modelos, formando lo que se denomina una capa del modelo. Muchos de los datos persistentes en una aplicación, almacenados en ficheros o bases de datos, deben residir en el modelo después de que los datos hayan sido cargados en la aplicación. Con ello se evita accesos de lectura y escritura innecesarios, aumentando la velocidad de respuesta de la aplicación. Debido a que los modelos representan el conocimiento y la experiencia en un dominio específico de un problema, estos pueden ser reutilizados en otros dominios similares. Idealmente, un modelo no tiene ninguna conexión explícita con la vista que presenta sus datos; no debe preocuparse por la interfaz de usuario y las cuestiones de presentación.

2) Interfaz: Las acciones del usuario en la vista que desencadenan la creación, modificación o destrucción de los datos se comunican a través de un objeto de control (el controlador) y dan lugar a la creación o actualización de un modelo. Cuando se produce un cambio en un modelo (por ejemplo, nuevos datos se reciben sobre una conexión de red), se notifica a un objeto controlador, que actualiza la vista apropiadamente.

B. La vista

1) Implementación: Un objeto de tipo vista es aquel mediante el cual los usuarios pueden interactuar. Un objeto de vista sabe cómo presentarse a sí mismo y puede responder a las acciones del usuario. Un propósito importante de los objetos de vista es mostrar los datos de los modelos de la aplicación y permitir la edición de esos datos. En una aplicación las vistas suelen ser desacopladas de los modelos. Los objetos de vista garantizan la coherencia entre las aplicaciones, ya que normalmente el código de las diferentes vistas suele ser reutilizable y reconfigurable.

2) Interfaz: Los objetos de vista aprenden acerca de los cambios en el modelo a través del controlador de la aplicación y comunican al modelo los cambios iniciados por el usuario, por ejemplo, el texto introducido en un campo de texto, a través del objeto controlador de una aplicación.

C. El controlador

1) Implementación: Un objeto controlador actúa como un intermediario entre uno o varios de los objetos de vista de una aplicación y uno o más de sus modelos. El controlador es un conducto a través del cual los objetos de vista aprenden acerca de los cambios en el modelo y viceversa. El controlador también suele ser el encargado de realizar la configuración y las tareas de coordinación de una aplicación y gestionar los ciclos de vida de otros objetos.

2) Interfaz: Un objeto controlador interpreta las acciones del usuario realizados en objetos de vista y comunica los datos nuevos o modificados a la capa del modelo. Cuando el modelo cambia, un objeto controlador comunica los nuevos datos del modelo a los objetos de vista de modo que la vista los pueda mostrar.

III. El paradigma de la programación orientada a objetos

Una vez se han comprendido los beneficios que aporta el diseñar una aplicación siguiendo un procedimiento de separación MVC se ha de elegir un lenguaje de programación que permita pasar del diseño MVC a código de forma fácil. Existen un gran numero de *frameworks* (término en inglés para definir al "entorno de trabajo" para el desarrollo de aplicaciones) que apoyándose en diferentes lenguajes siguen un procedimiento de diseño basado en MVC [4]. No es de extrañar que en la mayoría de estos *frameworks* se ha hecho uso de lenguajes basados en el paradigma de la programación orientada a objetos (POO). Por este motivo, se ha considerado que este trabajo debería presentar las ideas fundamentales que hay detrás de la POO, sin centrarse en un lenguaje en concreto. Para profundizar en los patrones de diseño para las aplicaciones web en el contexto de la orientación a objetos se recomienda la lectura de *Patterns of Enterprise Application Architecture* de Martin Fowler [3].

La POO usa el concepto de objeto, el cual se caracteriza por una serie de propiedades y métodos, con el objetivo de diseñar aplicaciones y programas informáticos. Por ejemplo, si se trata de modelar un reloj en un esquema de POO, se puede considerar que el reloj es el elemento principal que tiene una serie de características, como podrían ser el color, la representación digital o analógica o la marca. Además tiene una serie de funcionalidades asociadas, como pueden ser establecer la hora, la fecha o activar una alarma. Entonces, en un esquema POO el reloj sería el objeto, las propiedades serían las características y los métodos serían las funcionalidades asociadas.

En la POO también se suele hacer referencia a la interfaz e implementación de un objeto. Siguiendo con el ejemplo del reloj, ver figura 3, cuando se habla de implementación se estaría haciendo referencia a la mecánica interna del reloj y que permite a la interfaz mostrar la hora. Además, la interfaz puede permitir al usuario realizar ajustes en la implementación

del reloj, como puede ser el control mediante manecillas para ajustar la posición de las agujas.

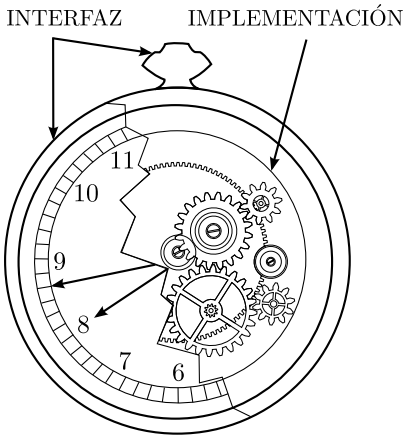

Fig. 3. Concepto de interfaz e implementación.

La POO es una forma de programar que trata de encontrar una solución a estos problemas. Introduce nuevos conceptos, que superan y amplían conceptos antiguos ya conocidos. A continuación se presentan una serie de técnicas inherentes en la POO.

A. Herencia

Es la técnica más conocida y más fácil de entender. Para explicarla, es necesario introducir el concepto de clase. Todo objeto que se instancia pertenece a una clase. La clase es una forma de otorgar ciertas propiedades y métodos comunes a los objetos pertenecientes a una misma clase. Las clases no están aisladas, sino que se relacionan formando categorías y subcategorías entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. De esta forma, para el ejemplo expuesto anteriormente del reloj, se podría extender el comportamiento de la clase reloj a las subclases reloj de pulsera y reloj de pared.

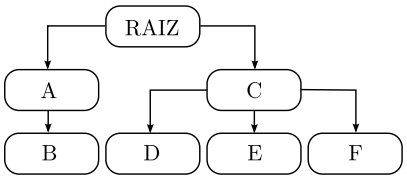

Fig. 4. Ejemplo de jerarquía de clases.

Como se puede ver en la figura 4, la herencia organiza y facilita el polimorfismo y el encapsulamiento permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir y extender su comportamiento sin tener que volver a implementarlo, lo que se conoce como sobrecarga de métodos.

B. Abstracción

Los objetos de una clase permiten al programador abstraerse de la implementación interna del mismo. Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado o propiedades, y "comunicarse" con otros objetos en el sistema sin revelar cómo se implementan estas características. El proceso de abstracción permite seleccionar las características relevantes dentro de un conjunto e identificar comportamientos comunes para definir nuevos tipos de entidades en el mundo real. La abstracción es clave en el proceso de análisis y diseño orientado a objetos, ya que mediante ella podemos llegar a armar un conjunto de clases que permitan modelar la realidad o el problema que se quiere atacar.

Siguiendo con el ejemplo de la figura 3 se puede ver que un usuario (programador) no necesita entender el funcionamiento interno de un reloj, o dicho de otra manera, su implementación, puesto que es la interfaz la que le permite interactuar de forma transparente con la estructura interna.

C. Polimorfismo

Puede ser considerada como una de las herramientas de programación mas versátiles dentro de la POO. Comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre, al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando. Cuando esto ocurre en "tiempo de ejecución", esta última característica se llama asignación dinámica de métodos. Algunos lenguajes proporcionan medios estáticos, en "tiempo de compilación", tales como las plantillas y la sobrecarga de operadores de C++.

D. Encapsulamiento

Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de los componentes del sistema. Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una interfaz a otros objetos que especifica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas, solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no pueden cambiar el estado interno de un objeto de maneras inesperadas, eliminando efectos secundarios e interacciones inesperadas.

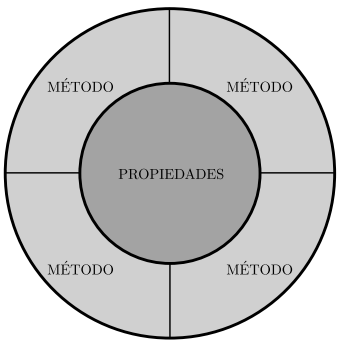

Fig. 5. Encapsulamiento de una clase.

IV. Sistema de gestión de contenidos

Gracias a las técnicas inherentes a cualquier lenguaje orientado a objetos y haciendo uso de la separación entre interfaz e implementación se puede implementar de forma eficaz y muy bien estructurado un *framework* basándose en

la separación MVC. En el diseño de web existen multitud de estas herramientas, siendo los mas conocidos los sistemas de gestión de contenidos (en inglés *Content Management System*, abreviado CMS).

Un CMS es un programa que permite crear un *framework* para la creación y administración de contenidos, principalmente en páginas web, por parte de los participantes. Consiste en una interfaz que controla una o varias bases de datos donde se aloja el contenido del sitio. El sistema permite manejar de manera independiente el contenido y el diseño. Así, es posible manejar el contenido y darle en cualquier momento un diseño distinto al sitio sin tener que darle formato al contenido de nuevo, además de permitir la fácil y controlada publicación en el sitio a varios editores. Esto permite gestionar, bajo un formato estandarizado, la información del servidor, reduciendo el tamaño de las páginas para descarga y reduciendo el coste de gestión del portal con respecto a una página estática, en la que cada cambio de diseño debe ser realizado en todas las páginas, de la misma forma que cada vez que se agrega contenido tiene que maquetarse una nueva página HTML y subirla al servidor web.

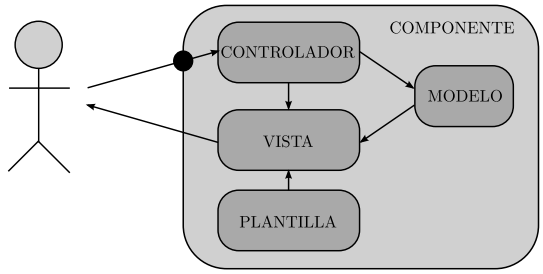

Fig. 6. Extensión del procedimiento MVC en un gestor de contenidos.

Para lograr esta separación entre los contenidos y el diseño, la mayor parte de los CMS hacen uso de una extensión conceptual del procedimiento MVC. En la figura 6, se muestra la filosofía de diseño planteada por uno de los CMS más usados en la red: *Joomla!* [5]. El punto de entrada es independiente del controlador ya que el gestor de contenidos toma el control de las acciones del usuario mediante esta entrada. Una vez realizadas ciertas comprobaciones de seguridad y preparar las variables de usuario y del sistema, toma el mando el controlador principal que determinará que vista y modelo cargar. Lo más importante de este diagrama es el concepto de separación de la plantilla de presentación del resto del flujo de ejecución. La plantilla se diseña de forma independiente a los contenidos y es gracias a una serie de llamadas al *framework* del CMS [6] que se puede hacer uso de dicho diseño desde el programa principal. De esta forma se independizan contenidos y diseño.

V. Aplicación práctica para el control de flotas heterogéneas

Sentadas las bases de diseño y con una idea más clara de la motivación detrás del procedimiento de programación se va a plantear de forma genérica el funcionamiento de un sistema para la gestión y control de flotas heterogéneas. Existen tres grandes grupos de flotas atendiendo al medio de transporte: terrestres, marítimas y aéreas.

Para agilizar los procesos de distribución y abarcar una mayor demanda del servicio prestado nace el control y gestión de las flotas mediante sistemas avanzados de comunicaciones. Estos sistemas de comunicaciones pueden estar basados en comunicaciones por satélite, comunicaciones móviles como pueden ser las ya consolidadas redes 3G haciendo uso de la tecnología HSDPA/HSUPA (acrónimo en inglés de *High Speed Downlink/Uplink Packet Access*) o las novedosas redes móviles 4G mediante la tecnología inalámbricas WiMax o la tecnología móvil LTE.

Independientemente de la tecnología de comunicaciones utilizada, este tipo de aplicaciones hacen uso de servicios de mensajería ya consolidados, como puede ser el servicio MHS (acrónimo en inglés de *Message Handling Service*) desarrollado por Novell a finales de la década de los 80. Estos servicios de mensajería usan formatos estandarizados para sus mensajes, SMF (*Standard Message Format*). Los terminales de comunicaciones que se embarcan en las flotas hacen uso de dichos estándares, de esta forma, y siguiendo el procedimiento de programación planteado en este trabajo, se puede implementar una única aplicación con una interfaz común para el control y gestión de flotas.

VI. Conclusiones

Las pautas de diseño planteadas en este trabajo pueden en principio resultar un tanto complejas y requieren de un esfuerzo inicial por parte de los desarrolladores que se involucran en la implementación de un sistema de gestión y control de flotas heterogéneas (sobre todo si no se tienen nociones sobre POO), pero la curva de aprendizaje de estos sistemas suele ser rápida y los beneficios son múltiples. El procedimiento de programación basado en la separación MVC da como resultado aplicaciones mas escalables, modulares y más fáciles de mantener.

Más aún, con los gestores de contenidos se puede llegar a lograr una aplicación que se extiende mas allá de las funcionalidades de la aplicación original sin la necesidad de un programador especializado, lo que reduce los costes que puede suponer reimplementar una aplicación; aunque para lograr este propósito las labores de diseño y programación del sistema se complican notablemente y requieren de un mayor tiempo de desarrollo.

Agradecimientos

Este trabajo forma parte de los resultados del proyecto de investigación financiado por el MCINN con referencia TEC 2009-14307-C02-02 y TECMUSA PSS-370000-2009-45/46/47.

Referencias

[1] http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html
[2] http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html
[3] Martin Fowler, *Patterns of Enterprise Application Architecture* Addison-Wesley, November 05, 2002.
[4] http://es.wikipedia.org/wiki/Modelo_Vista_Controlador#Frameworks_MVC
[5] Joseph L. LeBlanc, *Learning Joomla! 1.5 Extension Development,* 1st ed., Packt publishing Ltd., 2007.
[6] http://docs.joomla.org/