CENTRO DE ESTUDIOS DE POSTGRADO

Máster en Profesorado de Enseñanza Secundaria Obligatoria, Bachillerato, Formación Profesional y Enseñanza de Idiomas

UNIVERSIDAD DE JAÉN
Centro de Estudios de Postgrado

Trabajo Fin de Máster

MODELO-VISTA-CONTROLADOR. LENGUAJE UML

Alumno/a: **Alonso Aranda, Carlos**

Tutor/a: Prof. D. Pedro González García
Prof. Dª Carmen Martínez Cruz
Dpto: Informática

Junio, 2019

ÍNDICE

RESUMEN

Este trabajo está dividido en dos partes claramente diferenciadas:

Por un lado, se realiza un estudio epistemológico sobre el Modelo-Vista-Controlador y el lenguaje UML, analizando los antecedentes de ambas cuestiones, el estado del arte, así como las tendencias futuras relacionadas con los patrones en lenguajes de programación.

La segunda parte de este documento es la elaboración de una Unidad Didáctica, enmarcada dentro del Ciclo Superior de Desarrollo de Aplicaciones Web, cuyo objetivo es realizar una proyección didáctica de los contenidos desarrollados en la primera parte de este documento, así como su contextualización dentro de un centro. También se abordará la normativa asociada al módulo que lo contiene.

Palabras clave: Modelo-Vista-Controlador, MVC, patrones de lenguaje, UML, Programación Orientada a objetos, didáctica

ABSTRACT

This work is divided in two clearly differentiated parts:

On the one hand, an epistemological study about Model-View-Controller and UML language, analysing the antecedents of both issues, the state of the art, as well as the future trends related to the patterns in programming languages.

The second part of this document is the elaboration of a Didactic Unit, framed within the Higher Technical Certificate on Web Applications Development, whose objective is to carry out a didactic projection of the contents developed in the first part of this document, as well as its contextualization within a high school center. The regulations associated with the module that contains it will also be dealt with.

Keywords: Model-View-Controller, MVC, language patterns, UML, Object Oriented Programming, didactics

A. ESTUDIO EPISTEMOLÓGICO

1. INTRODUCCIÓN

Al igual que cuando un arquitecto diseña los planos de un edificio, la Arquitectura de Software es el diseño global de la estructura y la interacción entre las distintas partes de una aplicación. Cuando desarrollamos software, tenemos que planear la estructura general de nuestra aplicación para mejorar la organización, la sostenibilidad y la flexibilidad.

En líneas generales, no es necesario diseñar una arquitectura de software cada vez que construimos un sistema de información, ya que a día de hoy existen multitud de arquitecturas distintas, enormemente especializadas para cubrir una necesidad concreta. Cada una de estas arquitecturas tiene sus ventajas y desventajas, obviamente, pero han sido largamente probadas a lo largo de la corta historia de la informática y las nuevas tecnologías.

Existen arquitecturas clásicas muy conocidas, como la arquitectura cliente-servidor, o la descomposición modular, que prácticamente no necesitan presentación. Sin embargo, surgieron nuevos paradigmas debido al creciente aumento de actores implicados en la ejecución de un proyecto software, que demandaban mayor independencia, mantenimiento y reusabilidad en el código sobre el que se trabajaba. Aquí es donde el Modelo-Vista-Controlador presentó sus credenciales.

Por otro lado, los distintos paradigmas de desarrollo contienen un determinado número de vistas o modelos que los caracterizan, y todos tienen la necesidad de ser representados. La llegada del UML (Unified Modeling Lenguage) supuso una auténtica revolución en este sentido. Hasta entonces, cada metodología utilizaba una notación y semántica distinta, lo que generaba un cierto caos en la representación. UML vino a resolver esto desde su mismo nombre: Lenguaje Unificado de Modelado. Es tal su importancia actual, que prácticamente todos los lenguajes utilizan el UML como el estándar para mostrar sus modelos y vistas.

Por todos estos motivos es tan necesario abordar estas dos cuestiones tan importantes, porque configuran el pasado, presente y futuro del desarrollo software.

2. MODELO-VISTA-CONTROLADOR

2.1. ANTECEDENTES

El Modelo-Vista-Controlador surgió con los albores de las interfaces gráficas de usuario, para cubrir una necesidad creciente en el mundo del desarrollo de software en prácticamente todas sus variantes, la de agilizar y dar robustez al ciclo de vida del desarrollo, potenciando características hoy día muy interiorizadas, pero no así en los inicios del sector, tales como el mantenimiento de los procesos o la reutilización del código.

Coincidió en el tiempo, y no por casualidad, con el auge de la ingeniería del software, ya que esta se preocupaba de que los programas desarrollados cumpliesen unos estándares de calidad, marcando una serie de patrones deseables, justamente aquellos que el Modelo-Vista-Controlador pretendía satisfacer.

Esta rama de la informática, en su intento de mejorar los procesos de creación de software, potenciaron un concepto íntimamente ligado al Modelo-Vista-Controlador: la separación de conceptos, una estrategia de arquitectura de software basada en capas, en la que el código está agrupado en base a su responsabilidad sobre la solución propuesta [1].

Un ejemplo análogo para entender este paradigma lo tenemos en los inicios de Internet, cuando el código HTML era el responsable del 100% del código necesario para mostrar una página web, de modo que los conceptos de *backend* y *frontend* carecían de sentido, ya que la función de las etiquetas que usaba este lenguaje se mezclaban, independientemente de su funcionalidad. De este modo, por un lado disponíamos de etiquetas como *font*, utilizadas para definir las características propias de una fuente; por otro, etiquetas que servían para configurar atributos de la página web a mostrar, como podía ser *align*, cuya función es la de alinear horizontalmente los elementos asociados a dicha etiqueta. La practicidad de esta forma de agrupar código era más que cuestionable, ya que provocaba una ingente cantidad de modificaciones que debíamos propagar por todo el código cuando algún elemento se modificaba, y dicha modificación afectaba a la totalidad de la web. Para dar solución a esta problemática surgió el CSS (Cascading Style Sheets), en el que se separaban un lenguaje de otro en función de su responsabilidad [1].

Esta separación de código también surgió a consecuencia de la apertura de la programación a distintas clases de desarrolladores. Así, perfiles menos técnicos, como pueden ser los diseñadores, podían trabajar de manera más autónoma, sin preocuparse en exceso por un código que no dominaban [1]. De igual manera, los programadores podían desarrollar código abstrayéndose de la forma en la que los datos iban a ser mostrados por los diseñadores.

Lo anteriormente citado no dejan de ser meros ejemplos para contextualizar una época en la que la necesidad de una arquitectura similar al Modelo-Vista-Controlador era fundamental, ya que con la reutilización de código, que por experiencia sabemos que cumple la necesidad para la que fue creado, conseguimos agilizar el desarrollo y, por ende, obtener software de mayor calidad [2].

2.2. Definición y características

El Modelo-Vista-Controlador fue diseñado por primera vez por Trygve Reenskaug, en Smalltalk-76, un lenguaje de programación reflexivo, orientado a objetos y con tipado dinámico, durante la visita de este a Xerox Park en la década de los 70 [3]. Unos años más tarde, durante la década de los 80, un grupo de desarrolladores, entre los que se encontraba Jim Althoff, evolucionaron el concepto para la biblioteca de clases de la versión 80 de Smalltalk [4]. El salto al resto de lenguajes de programación como algo aceptado y respetado se produjo en 1987, a raíz de un artículo titulado: '*Applications Programming in Smalltalk-80™: How to use Model-View-Controller (MVC)*' [5], en el que se sentaban las bases de la aplicación de dicha arquitectura de una manera más general.

Hablamos de una arquitectura presentada incluso antes de la aparición de la web [1], como una solución sencilla y potente para poner en práctica la separación entre el *frontend* y el *backend* de una aplicación. Aunque surgió durante el desarrollo de Smalltalk, en poco tiempo se convirtió en la arquitectura estándar para el resto de lenguajes de programación orientados a objetos, sobre todo en aquellos desarrollos donde primaba el uso de interfaces de usuario, ya que su funcionamiento se fundamentaba en la separación del código en tres capas distintas, autónomas, y cada una con una responsabilidad definida, a las que llamaron Modelos, Vistas y Controladores (*Model, Views & Controllers*, por sus siglas en inglés) [1].

Más adelante profundizaremos en las características de cada una de estas capas. No obstante, en la primera propuesta de la arquitectura se definían las distintas capas de una forma bastante sencilla. Por ejemplo, el controlador se trataba del '*módulo que se encarga de la entrada*', mientras que la vista era '*el módulo que se encargaba de la salida*' [6]. Lógicamente, esta definición simplista fue evolucionando con el desarrollo de aplicaciones modernas, en las que las distintas responsabilidades se reparten entre la 'vista', y un framework específico asociado al lenguaje de programación en el que desarrollemos, mientras que el 'controlador' se convierte en un módulo a medio camino que hace las veces de intermediario en la comunicación entre las dos capas restantes. Esta evolución es fruto del éxito de la arquitectura.

La separación de conceptos que hemos comentado anteriormente, permite que las diferentes partes que componen la aplicación se puedan separar en módulos independientes, facilitando el mantenimiento posterior [2], así como la reusabilidad del código, dos tendencias que garantizan la calidad del código desarrollado.

En la siguiente imagen, podemos ver un esquema del Modelo-Vista-Controlador:

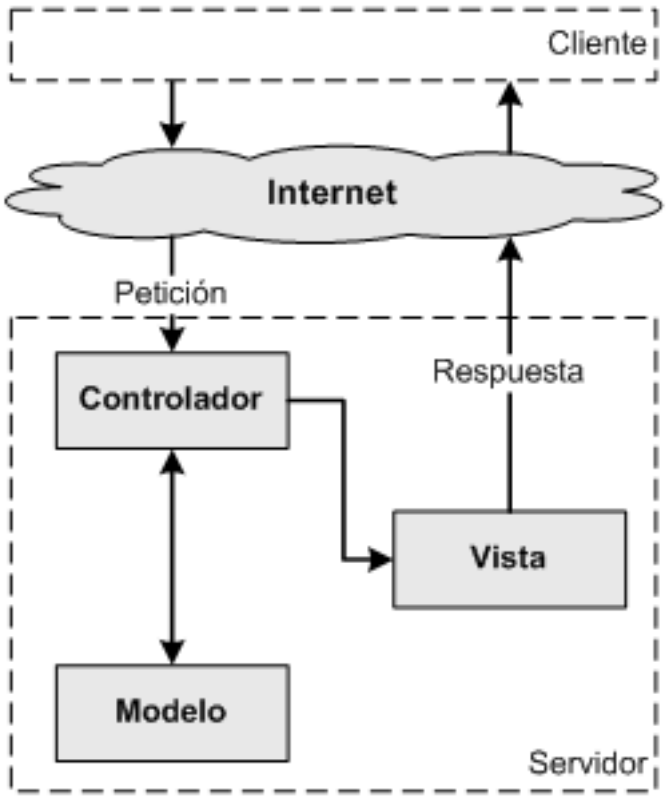

FIGURA 1: ESQUEMA MODELO-VISTA-CONTROLADOR[1]

Tal y como podemos observar en la Figura 1, si, por ejemplo, necesitamos cambiar algo en la vista, esa petición no se hace de forma directa al modelo, sino que es el controlador quien la gestiona, de modo que cualquier cambio bidireccional es transparente. Los cambios que se produzcan en cada módulo quedarán restringidos a este, lo que garantiza un desarrollo robusto, y facilita la gestión y control de errores.

Una vez definidas unas pautas introductorias sobre la arquitectura, pasamos a desglosar las partes o conceptos que lo forman.

[1] Figura obtenida de: http://www.juntadeandalucia.es/servicios/madeja/sites/default/files/imagecache/wysiwyg_imageupload_big/wysiwyg_imageupload/10/MVC_0.png

2.3. MODELO

Se trata de la parte del sistema que maneja directamente los datos, es decir, el que realiza las operaciones para obtener los resultados, por tanto contendrá aquellos submódulos/clases necesarios para acceder, mostrar o refrescar dicha información. En el diseño más común de una aplicación, los datos estarán alojados en una base de datos externa, de modo que en el modelo se definirán las conexiones necesarias del sistema con el Sistema Gestor de Base de datos (en adelante, SGBD), usualmente con una capa de acceso a datos, si se utiliza el concepto de capas. En cualquier caso, estos datos 'externos' no pertenecen al modelo.

No obstante, lo habitual cuando se trabaja con esta arquitectura es usar ciertas librerías que permitan abstraernos del SGBD en el que estén alojados los datos, de modo que en lugar de lanzar directamente las sentencias SQL, cuya ejecución es costosa, los datos se manipulen a partir de llamadas a las sentencias encapsuladas en un serie de clases y objetos que utilizarán un dialecto que el SGBD sepa entender [1]. De este modo, no sólo el modelo es independiente del resto de objetos de la arquitectura, sino que los propios datos son independientes del modelo.

Por tanto, el modelo se encargará de [7]:

- Acceder a los datos, así como manipularlos según la necesidad.
- Establecer la funcionalidad del sistema. A esto se le llama las '*reglas de negocio*'. Sobre este aspecto profundizaremos más adelante, no obstante, un ejemplo de regla de negocio podría ser: «*Si el cliente devuelve el producto en los primeros 15 días tras la compra, la devolución será gratuita*».
- Llevar un conteo de las vistas y controladores del sistema.
- En el caso de que trabajemos con un modelo activo, deberá notificar a las vistas aquellos cambios en los datos que puedan ser provocados por un agente externo (un disparador que provoca una actualización, etc).

En su primera definición, se teorizó sobre que el modelo no debería siquiera tener constancia de la existencia de la vista y del controlador. No obstante, la aplicación de esta característica es difícil, pues debe existir algún tipo de interfaz que permita a los distintos módulos comunicarse entre sí. Así, la arquitectura propuesta en Smalltalk sugería que el modelo se subdividiese en dos módulos de aplicación más práctica: el modelo del dominio y el modelo de la aplicación [8].

Se entiende por modelo de dominio al conjunto de clases asociadas al análisis del problema real, a raíz del cual queremos buscar una solución a partir de nuestro desarrollo, de modo que no debería estar relacionado en ningún caso con algún condicionante externo distinto a los datos que contiene.

Por otro lado, el modelo de la aplicación, llamado también '*coordinador de la aplicación*' es aquel que se encarga de establecer la comunicación entre las vistas y el modelo del dominio. Así, tiene conocimiento de estas, e incorporan los módulos necesarios para notificar a éstas los cambios que se pudieren dar en el modelo del dominio.

2.4. VISTA

Las vistas, como podemos deducir, contienen aquellos módulos que se van a encargar de materializar las interfaces de usuario de nuestra aplicación, de modo que siempre mostrarán la información más actualizada, así el programador puede despreocuparse de ellas, ya que es el módulo quien se encarga automáticamente de dicha actualización.

De igual manera que el modelo hacía referencia al *backend* de nuestra aplicación, las vistas hacen referencia el *frontend*; es decir, se encargan de definir la interfaz de usuario, en el caso de una aplicación de escritorio, o la apariencia de una página web, en el caso que aplique [9].

Las vistas también se encargan de mostrar los datos, pero no de acceder directamente a ellos. De eso se encarga el Modelo, de modo que serán las vistas quienes hagan una petición de los datos al Modelo, y a partir de ellos mostrarán la salida al usuario. Para ello se dispone normalmente de un controlador, el cual la vista instancia [7].

La relación entre los datos y las vistas que los muestran es de uno a muchos [8]. Es decir, el mismo dato, asociado al Modelo, puede ser representado de distinta manera en la misma instancia. Así, por ejemplo, podemos tener una vista que muestra la batería restante con una imagen de una barra, o directamente mostrando el porcentaje. La información en sí no cambia, únicamente cambia lo que se proyecta al usuario.

2.5. CONTROLADOR

Por último, pero no menos importante, el controlador es el enlace imprescindible entre las vistas y los modelos, de modo que sirve para dar respuesta a la comunicación bidireccional entre ambos elementos. Sin embargo, ni se encarga de manipular directamente los datos, responsabilidad del Modelo; ni de definir la interfaz de usuario en el que estos van a ser mostrados, tarea de la vista. Únicamente hace de puente, establecimiento los mecanismos y reglas para responder a la dinámica de la comunicación [1].

Así, el controlador tiene la implementación del código necesario para dar respuesta a las peticiones que llegan desde la aplicación, que normalmente vienen disparadas por el usuario, como pueden ser mostrar u ocultar una barra de tareas, pinchar en cualquier enlace de una aplicación web, dar respuesta a cualquier botón de una aplicación de escritorio, etc.

El controlador debe ser capaz de redirigir una petición, ya que esta puede ser atendida por las vistas o por los modelos, según la necesidad. Por tanto, es también quien se encarga de atender esos eventos de entrada, y definir las reglas que acotan y dirigen esos eventos al módulo que debe resolverlos. Profundizaremos en este aspecto en el siguiente apartado.

2.6. OTROS CONCEPTOS ASOCIADOS

Existen una serie de conceptos que no están específicamente asociados al Modelo-Vista-Controlador, aunque en este cobra especial relevancia. Es el caso de la '*lógica de negocio*', que se trata del conjunto de reglas implementadas en nuestro software, con el objetivo de dar respuesta a las distintas situaciones que puedan producirse. Cuando un usuario interacciona con la interfaz gestionada por la vista, acciona un disparador cuya funcionalidad está determinada por la '*lógica de negocio*'.

Además de eso, en la lógica de negocio vienen recogidas y acotadas todas aquellas normas que definen lo que un usuario puede hacer y no hacer [1]. Es decir, se encargan de definir aquellas reglas de comportamiento de la aplicación durante la ejecución de la aplicación. En el caso que nos ocupa, son los modelos quienes se encargan de esta tarea, definiendo qué respuesta dar a esas situaciones, así como permitir qué puede o no ocurrir en tiempo de ejecución.

Imaginemos por ejemplo que tenemos la arquitectura montada en un servicio web de compra, en el que debemos establecer qué hacer cuando un usuario añade un producto al carrito. En la lógica de negocio debe venir implementada de manera clara cómo dar respuesta a todas las situaciones que pueden ocurrir durante el proceso de tramitación. Pero no únicamente de la compra, sino de la gestión del usuario, el timeout del pedido, las pasarelas de pago, etc. Además, se pueden definir distintos roles en función del tipo de usuario, de si ya está registrado, de si dispone de algún tipo de ventaja premium, etc. Toda esta responsabilidad es parte del modelo.

Además de la lógica de negocio, existe otro concepto asociado a la responsabilidad del controlador, que es la '*lógica de la aplicación*'. Esta se define como el conjunto de gestiones que realiza el controlador para satisfacer una petición [1], gestionando la comunicación entre las vistas y el modelo, de modo que las primeras puedan recibir los

datos actualizados cuando correspondan, y los métodos de los modelos sean invocados para dar respuesta a una petición.

Por ejemplo, siguiendo con el ejemplo que comentábamos en el apartado anterior, es posible que el modelo tenga unos métodos u otros en función del tipo de usuario que realice la petición, de modo que tendremos que establecer una serie de reglas que definan qué módulos de los modelos deben ser llamadas en función de unos parámetros de entrada. Si el usuario no está registrado, será necesario realizar una serie de pasos que deberán estar contemplados en la lógica de la aplicación. Por otro lado, si el usuario está registrado pero no es premium, se satisfará la lógica de una forma, y de otra si sí lo es. Además de la llamada a los módulos, deberá gestionar qué vistas se muestran, de qué manera, etc. En definitiva, todo esto debe estar contemplado por el controlador.

Esta lógica de aplicación tampoco es algo específico del Modelo-Vista-Controlador, pero es fundamental para entender dónde empieza y acaba la responsabilidad de cada elemento de nuestro sistema, de modo que podamos organizar el código con coherencia, y así hacerlo más mantenible y reusable.

2.7. Funcionamiento de la arquitectura MVC

Una vez definidos los elementos que componen este patrón de arquitectura, haremos un repaso de cómo sería un flujo de trabajo característico en este esquema [7]:

1. El usuario interactúa de alguna forma con la vista, es decir, con la interfaz de usuario, por ejemplo pulsando cualquier botón de nuestra aplicación. Esa solicitud le llega al controlador, que se encargará de gestionarla de ahora en adelante hasta que esté resuelta.
2. El controlador gestiona el evento que se acaba de disparar, normalmente a través de un gestor de eventos, y comienza la comunicación tanto con los modelos como con las vistas. Se consulta la lógica de la aplicación y la lógica de negocio, de modo que se solicitará a los modelos los datos que sean necesarios, mientras que haremos la petición a la vista de cómo deben ser mostrados.
3. El controlador se comunica con el modelo, actualizándolo siempre, y realizando las modificaciones que el usuario, a través de su petición, haya solicitado.
4. El controlador, con los datos obtenidos de los modelos, se encargará de enviarlos a las vistas, haciendo de nexo de comunicación. Sería corriente tanto una cosa como la otra, todo depende de nuestra implementación.
5. Las vistas muestran la salida al usuario.

6. La interfaz de usuario se pone a la espera de nuevas solicitudes por parte del usuario, reiniciando el ciclo.

Este flujo tiene multitud de ventajas, por ejemplo, la definición tan clara de la responsabilidad de cada módulo de la aplicación nos permite optimizar el código, favoreciendo la escalabilidad de la aplicación, fundamental para permitir su crecimiento exponencial [2]. Cuando se dispone de una base organizada y una lógica de aplicación robusta, se facilita enormemente la posibilidad de añadir nuevas funcionalidades a la aplicación.

Además, ha quedado demostrada su permeabilidad, ya que aunque originalmente fue diseñado para aplicaciones de escritorio, actualmente ha sido adaptada como arquitectura de cabecera en la implementación y diseño de aplicaciones web en multitud de lenguajes de programación. Existe prácticamente un framework desarrollado para implementar este patrón en cualquier lenguaje.

Esta adaptación del patrón en los frameworks también ha ido evolucionando, de un enfoque en el que prácticamente la totalidad de la carga recaía en el servidor, a un estado actual en el que los framework más recientes permiten la ejecución parcial de ciertos componentes en el cliente [8]. Todos estos hechos han desmontado aquellas voces que decían que era un patrón de diseño costoso y de difícil aplicación en un lenguaje que no siguiese el paradigma de diseño orientado a objetos.

Precisamente, una de las ventajas principales del patrón se convierte en una pequeña desventaja. Hablamos de la facilidad de mantenimiento. Si usamos este patrón de desarrollo, será necesario realizar un mayor esfuerzo durante las fases iniciales del desarrollo, pero es una inversión de retorno asegurado, ya que precisamente el mantenimiento suele ser lo más costoso en una aplicación de largo recorrido [8].

2.8. Variaciones del modelo

Dado su largo recorrido, a lo largo del tiempo han ido surgiendo distintas variantes, de modo que cada una potencia ciertos aspectos de la arquitectura. No es objeto de este trabajo profundizar en cada una de las variantes, pero sí es interesante esbozar alguna de las más usadas.

2.8.1. Modelo-Vista-Controlador Jerárquico

Se trata de una evolución del patrón original, adaptado para su utilización en la mayoría de aplicaciones web actuales, con el objetivo de mejorar la viabilidad de su utilización. Su origen lo tenemos en Julio del año 2000, en el sitio web JavaWorld [10], quien propuso que los tres elementos que componen este patrón se agruparan y se

subdividieran de manera jerárquica, de modo que se especializaran y se comunicaran entre ellos a partir de su controlador, quien redirige las peticiones a nivel interno, pero funcionando de manera independiente unos de otros, tal y como viene representada en la Figura 2.

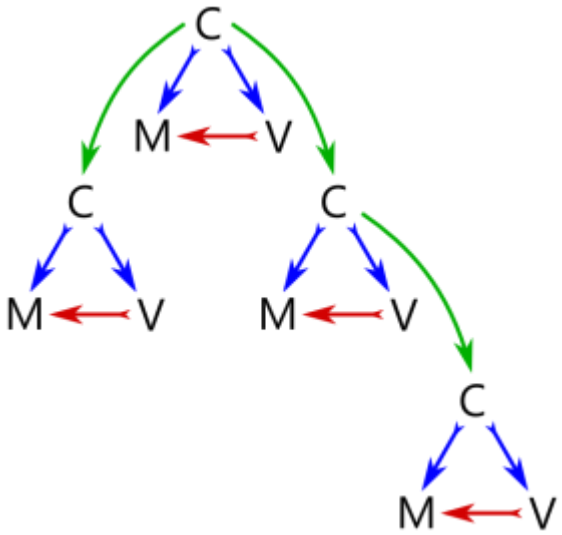

FIGURA 2: MODELO-VISTA-CONTROLADOR JERÁRQUICO[2]

Este esquema otorga múltiples ventajas:

- Se disminuyen las dependencias entre cada una de las partes que componen la aplicación.
- La independencia garantiza la reutilización.
- Aumenta la escalabilidad sin que el mantenimiento se vea resentido.

2.8.2. MODELO-VISTA-ADAPTADOR

El objetivo de esta variante es independizar el modelo de datos de la vista, de modo que los cambios en uno no afecten al otro, y viceversa.

La diferencia básica con el Modelo-Vista-Controlador original, es que este permite la comunicación entre la vista y el modelo, mientras que el Modelo-Vista-Adaptador '*obliga*' a que la comunicación entre ambos elementos se produzca a través de un pseudo-controlador al que llamamos '*adaptador*' [11].

La ventaja fundamental que proporciona es que ambos elementos se diseñan e implementan de manera completamente independiente, de modo que cualquier

[2] Figura obtenida de: http://www.hardycodes.com/wp-content/uploads/2018/09/640px-HMVCsvg.png

adaptación necesaria para la comunicación será implementada en el adaptador, sin necesidad de modificar ni el modelo ni las vistas.

2.8.3. MODELO-VISTA-PRESENTADOR

En esta variante se introduce un nuevo elemento, en sustitución del controlador, al que llamamos '*Presentador*'. Su función es la de recuperar datos del modelo y presentarlos a la vista ya formateados, de forma que permite a la vista ser totalmente pasiva, estando esta en muchos casos libre de toda lógica, delegando la responsabilidad en el presentador. Esta es su principal ventaja, convertir a la vista en un elemento '*tonto*' que se limita a mostrar los datos recibidos del presentador.

Se usa fundamentalmente en aplicaciones web, ya que permite que la vista se ejecute en el cliente, además de necesitar sensiblemente menos código que el Modelo-Vista-Controlador.

2.8.4. MODELO-VISTA VISTAMODELO

La característica principal de esta variante es que trata de independizar al máximo la interfaz de la aplicación de la lógica de esta, con el objetivo de facilitar el desarrollo de aplicaciones de escritorio más completas, más centradas en potenciar la interfaz de usuario.

En esta arquitectura, las responsabilidades del modelo y de la vista son similares a las de arquitectura original. Sin embargo, se introduce un nuevo elemento, en sustitución del controlador: el '*modelo de la vista*', que como su nombre indica, hace de intermediario entre el modelo y la vista, cumpliendo dos funciones principalmente:

- Contener la lógica de la representación.
- Independizar la interfaz.

La ventaja fundamental que proporciona es que permite a programadores y diseñadores trabajar a la vez, además de ser muy sencillo realizar pruebas unitarias sobre el modelo de la vista, proporcionando código más testable y reusable.

3. EL LENGUAJE UML

3.1. ANTECEDENTES

Gracias a la democratización de los computadores y al auge del desarrollo informático empresarial, han surgido innumerables métodos y notaciones para el diseño orientado a objetos. Alguno de ellos son HDM, RMM, OOHDM, EORM, etc [12]. Sin embargo, ninguno ha cosechado tanto éxito ni ha conseguido penetrar en la informática moderna como el lenguaje UML, ya que gracias a él los diseñadores únicamente tienen que aprenderse una notación para cubrir todas las fases en el diseño y construcción de sus aplicaciones.

El principal problema que tenían todos sus precursores es que cada uno de ellos estaba especializado en una parcela concreta del problema que pretendían solucionar, y además, y no menos importante, ninguno tenía detrás las privilegiadas mentes de los '*tres amigos*': Grady Booch, James Rumbaugh e Ivar Jacobson, y un patrón líder e innovador como la Rational Software Corporation.

Así se escribe la historia, la Rational Software Corporation se convirtió en el referente de la década de los 90 cuando consiguió incorporar a sus filas a los promotores de los dos esquemas de modelado orientado a objetos con más proyección de la época: James Rumbaugh, creador del Object-Modeling Technique, especializado en el análisis orientado a objetos; y a Grady Booch, creador del método del mismo nombre, especializado a su vez en el diseño orientado a objetos. Faltaba Ivar Jacobson, el creador de la ingeniería del software orientada a objetos, el cual se unió al proyecto un año más tarde [13].

Se trataba de tres puristas a nivel teórico y metodológico, que a menudo chocaban en la manera de entender y defender la parcela que cada uno defendía y dominaba. Así se ganaron el sobrenombre de '*Los tres amigos*'. Cada metodología tenía su potencial, pero la asociación de todas ellas las mejoró tanto individualmente como colectivamente, ya que consiguieron el objetivo final que todas compartían. Rational supo ver el potencial de este trabajo conjunto, de modo que les encargó la creación de un '*lenguaje unificado de modelado*'. Además, supieron entender que el producto que estaban creando era patrimonio del propio ecosistema, por lo que durante el congreso OOPSLA (Object-Oriented Programming, Systems, Languages, and Applications) del año 1996 consultaron con las compañías competidoras, las cuales propusieron cambios menores que finalmente fueron implementados [13]. De hecho, se creó un consorcio internacional llamado UML Partners en el que todos los participantes completaron las especificaciones de UML. Esta sensación de que todos fueron partícipes en la creación fue fundamental para el éxito del UML en sus inicios.

El esfuerzo conjunto de todos ellos derivó en la materialización de las versiones 0.9 y 0.91 en 1996, auspiciadas bajo el mecenazgo de empresas tan potentes como IBM, Oracle o Microsoft [14], ya que todas ellas supieron entender la importancia del producto que avalaban para su éxito.

La versión 1.0 fue liberada en Enero de 1997 [13], y pocos meses más tarde, tras completar las semánticas de la especificación, se presentó la versión 1.1, que finalmente fue aceptada por el OMG (Object Management Group), lo que fue el espaldarazo definitivo para convertirse en un estandarte en sistemas de todo tipo: bancos, transporte, comunicaciones, etc [13].

La principal influencia provino de la Object-Modeling Technique de Rumbaugh, por ejemplo en la notación de rectángulos para objetos y clases, aunque se incorporaron numerosas capacidades de los otros dos esquemas para especificar ciertos detalles de diseño. No obstante, desde el principio idearon que el ámbito del UML no podía acotarse únicamente a los Lenguajes Orientados a Objetos, sino que su uso podría adaptarse perfectamente a otros sistemas orientados a la ingeniería o las comunicaciones, a costa de perder esa especialización.

Desde sus primeras versiones, UML ha ido progresando considerablemente, incorporando revisiones menores prácticamente cada 2 años [15], siempre bajo la supervisión del OMG, así como una Guía de usuario redactada por '*Los tres amigos*' [16], que sigue siendo la referencia ineludible sobre la cuestión.

3.2. Definición y características

El lenguaje unificado de modelado (UML, del inglés Unified Modeling Language), es el lenguaje de modelado con más reconocimiento de la informática moderna, bajo la tutela del Object Management Group (OMG). Decimos que es un '*lenguaje*' porque sigue una serie de normas y una metodología [17], orientada a unificar la interpretación que todos los implicados en el diseño, desarrollo o arquitectura de un sistema (software o no) tienen sobre el propio sistema. La clave está en la interiorización de los símbolos propuestos, y la democratización de estos. Por tanto, es un lenguaje que va de lo general a lo particular. En este sentido, es utilizado tanto en la definición del sistema, como de cada uno de los componentes o métodos que lo forman, usado para documentar o como acompañante imprescindible en el desarrollo.

La potencia de su representación permite describir los sistemas tanto desde un punto conceptual, como pueden ser código asociado a procesos, etc; como a expresiones relacionadas con lenguaje de programación o bases de datos. Además, en su concepción, está totalmente deslocalizado de la metodología o lenguaje sobre la que trabajen los distintos diagramas, motivo por el cual puede ser usado por gran cantidad de perfiles distintos dentro de la organización [17]. No obstante, no podemos obviar que fue ideado como complemento a los lenguajes orientados a objetos, aunque su versatilidad lo ha convertido en el estándar del resto de lenguajes.

Como hemos comentado previamente, UML no es un lenguaje de programación, sin embargo existen multitud de herramientas que, a partir de los diagramas UML que modelemos, se pueden usar para generar código automáticamente [14], lo que demuestra la potencia y el avance actual del lenguaje de modelado.

3.3. TIPOS DE DIAGRAMAS UML

UML cuenta con varios tipos de diagramas, los cuales podemos dividir en dos subgrupos: diagramas *estructurales* y diagramas de *comportamiento*. En un primer acercamiento, la diferencia fundamental entre los dos tipos es que los segundos representan conceptos asociados a las interacciones entre los objetos, mientras que los primeros tienen un carácter estático. En un ejercicio de autoproyección, estos diagramas están jerarquizados, tal y como se muestra en la Figura 3.

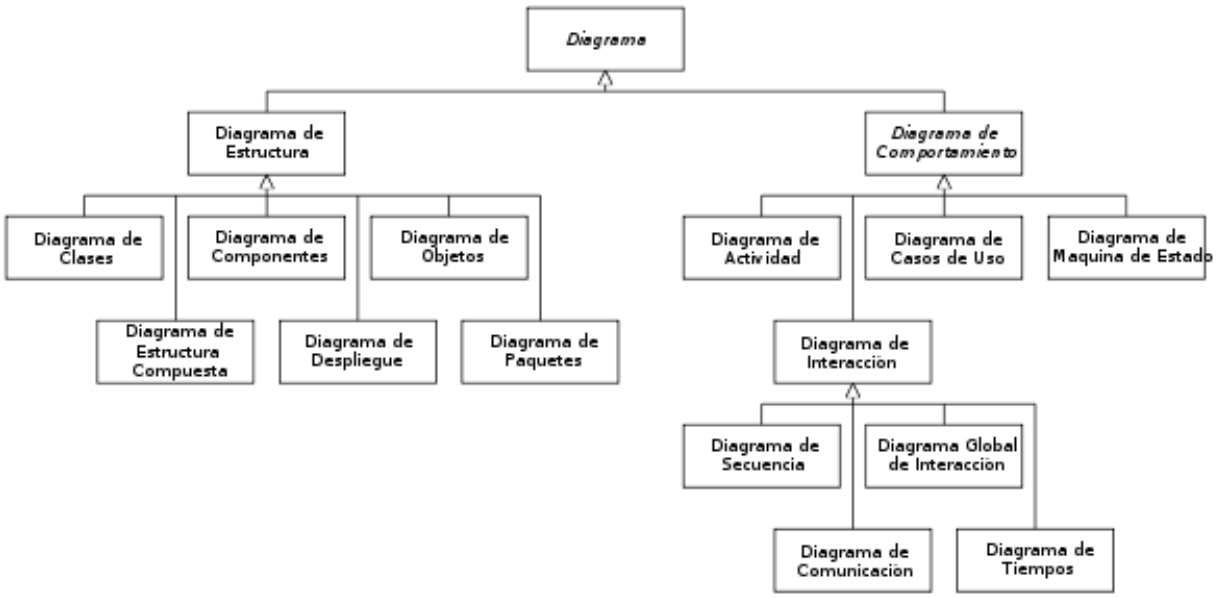

FIGURA 3: TIPOS DE DIAGRAMAS UML[3]

[3] Figura obtenida de: https://commons.wikimedia.org/wiki/File:Uml_diagram-es.svg

3.3.1. DIAGRAMAS ESTRUCTURALES

Se usan para mostrar la estructura estática de los elementos de un sistema. A su vez, se subdividen en:

- **Diagrama de clases:** El diagrama, sin ningún género de duda, más extendido de todos los que existen. Su nombre viene de su clara orientación a los lenguajes orientados a objetos, de modo que con este diagrama se pretende representar las clases de una aplicación, así como los métodos y atributos asociadas a estas, y lógicamente, la manera en que las clases se relacionan entre sí, simbolizadas por las flechas. Cada objeto lleva su nombre encima, a continuación los atributos de esa clase y por último los métodos que contiene.

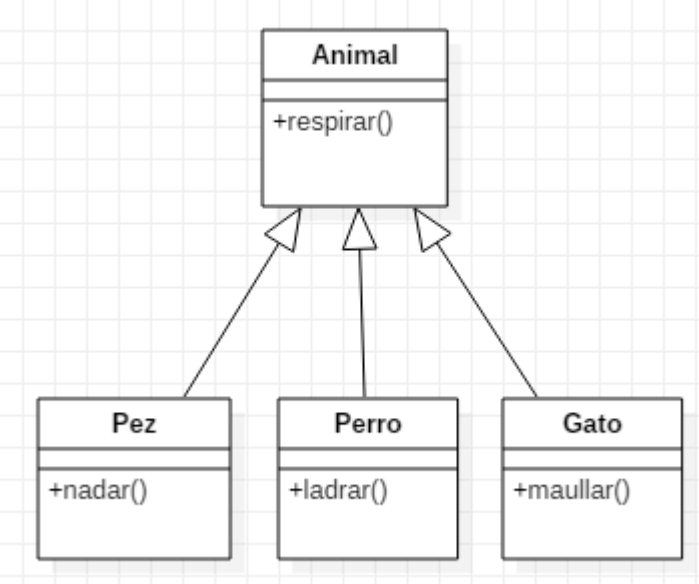

FIGURA 4: DIAGRAMA DE CLASES[4]

[4] Figura obtenida de: https://diagramasuml.com/wp-content/uploads/2018/08/clases6.png

- **Diagrama de componentes:** Es diagrama a más alto nivel, usado en sistemas cuya complejidad impide la representación a través de un diagrama de clases. Así, los objetos se relacionan entre sí mediante interfaces, que a su vez son enlazadas a través de conectores.

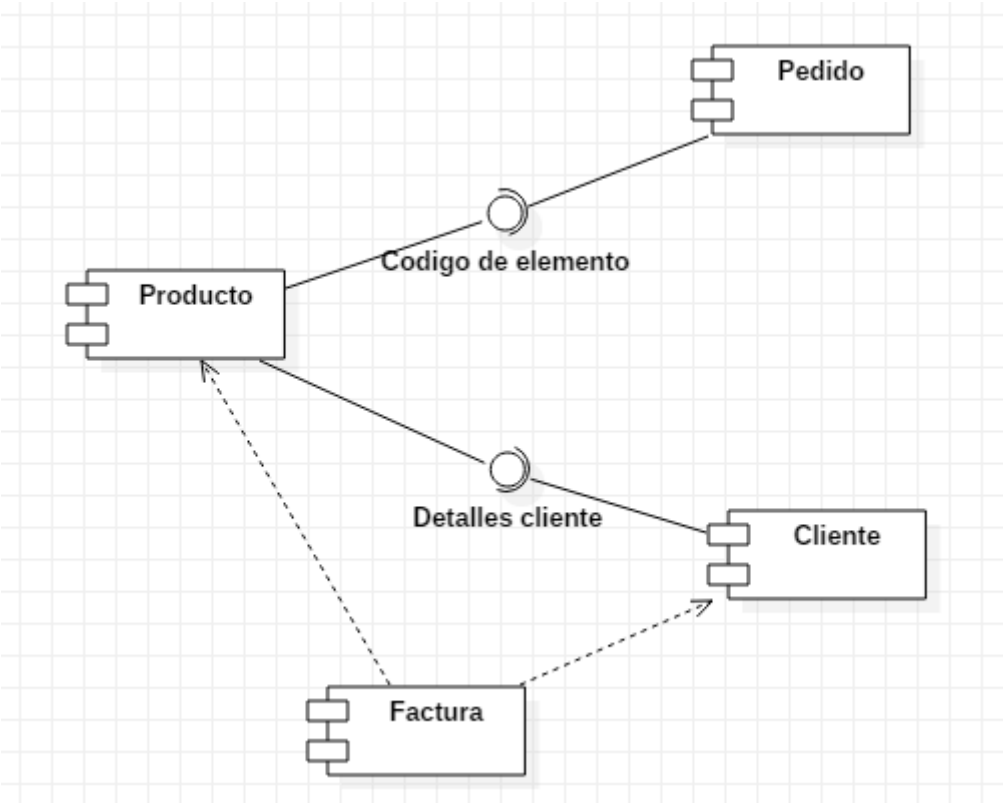

FIGURA 5: DIAGRAMA DE COMPONENTES[5]

[5] Figura obtenida de: https://diagramasuml.com/wp-content/uploads/2018/12/5.png

- **Diagrama de despliegue:** Es un tipo de diagrama orientado a ser usado cuando la implementación se despliega en varios equipos, cada uno de ellos con su configuración y características, ya que permite en un mismo diagrama mostrar todos los elementos software y hardware de un sistema.

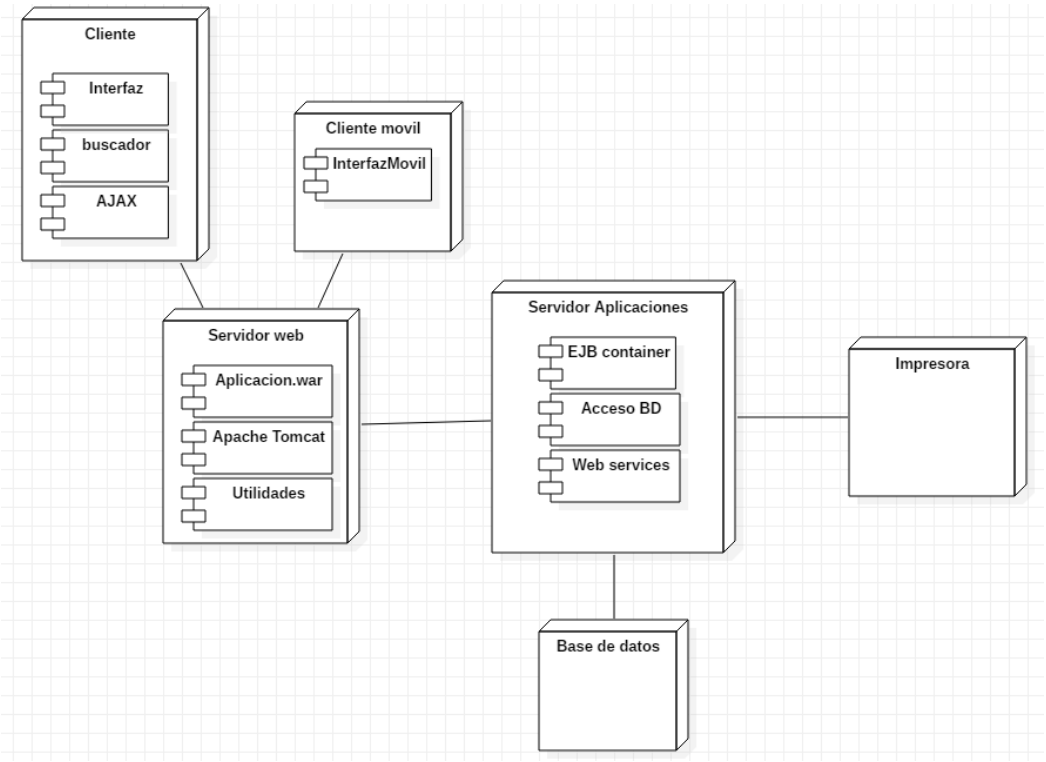

FIGURA 6: DIAGRAMA DE DESPLIEGUE[6]

- **Diagrama de objetos:** Son una especialización de un diagrama de clases, orientados a exponer relaciones complejas entre distintos objetos, a menudo usando ejemplos del mundo real, para hacer más comprensible dicho nexo. Debido a que a la hora de representarlos, se dispone de los datos asociados, se pueden usar para mostrar el estado de un sistema en un momento determinado.

FIGURA 7: DIAGRAMA DE OBJETOS[7]

[6] Figura obtenida de: https://diagramasuml.com/wp-content/uploads/2018/11/despliegue-1.png
[7] Figura obtenida de: https://diagramasuml.com/wp-content/uploads/2018/11/obj3.png

- **Diagrama de paquetes:** Se usan para representar de forma estática las dependencias entre los distintos paquetes de un sistema, a nivel lógico.

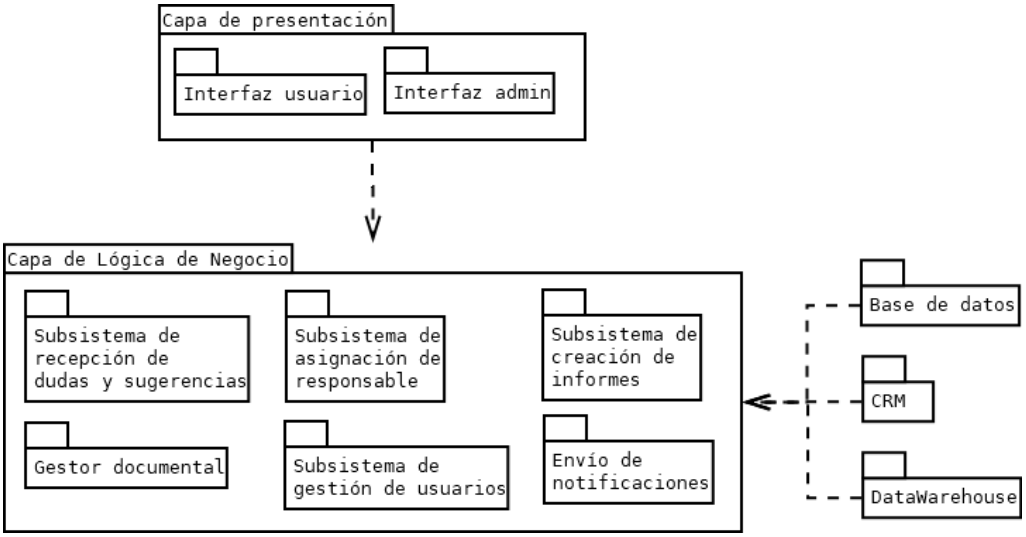

FIGURA 8: DIAGRAMA DE PAQUETES[8]

- **Diagrama de perfiles:** Es un tipo de diagrama que vino implementado en la versión 2.0 de UML, utilizado para generar diagramas especializados y orientados a un entorno concreto.

- **Diagrama de estructura compuesta:** Es también un diagrama relativamente reciente, y no muy extendido, usado para representar las relaciones entre clases a más bajo nivel, profundizando en el detalle.

3.3.2. DIAGRAMAS DE COMPORTAMIENTO

A diferencia de los diagramas estructurales, estáticos por naturales, los diagramas de comportamiento permiten mostrar las relaciones dinámicas entre los objetos de un sistema. Se subdividen en los siguientes tipos:

- **Diagrama de actividades:** Se usan para mostrar gráficamente los workflows de un sistema, tanto a nivel global, como a más bajo nivel, entrando en el detalle de algún componente específico de este. Uno de los usos más comunes es para mostrar cómo va cambiando de estado un sistema durante el transcurso de su ejecución.

[8] Figura obtenida de: https://diagramasuml.com/wp-content/uploads/2018/08/psq15.png

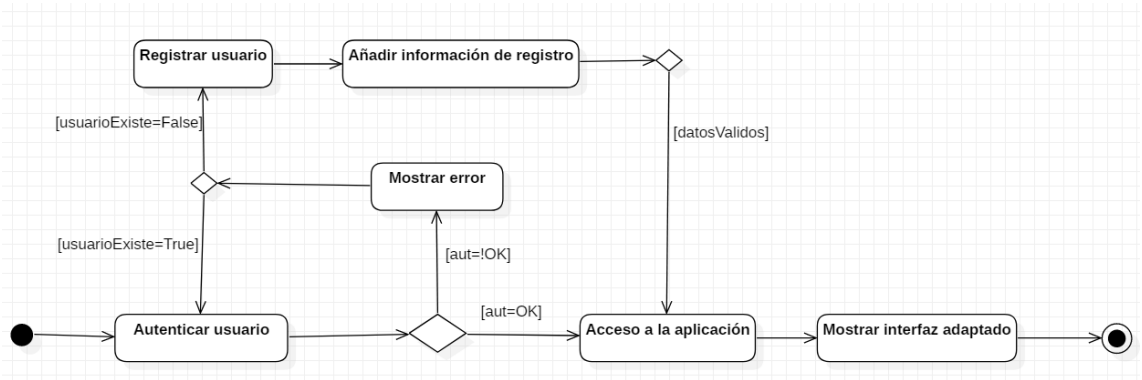

FIGURA 9: DIAGRAMA DE ACTIVIDADES[9]

- **Diagrama de casos de uso:** Es probablemente el diagrama de comportamiento más utilizado y democratizado, ya que permite mostrar una visión global de todos los componentes de un sistema, así como las responsabilidades de todas las partes implicadas y cómo se relacionan esas responsabilidades. Suele usarse en fases iniciales de un proyecto para determinar el alcance de este, así como identificar los componentes necesarios.

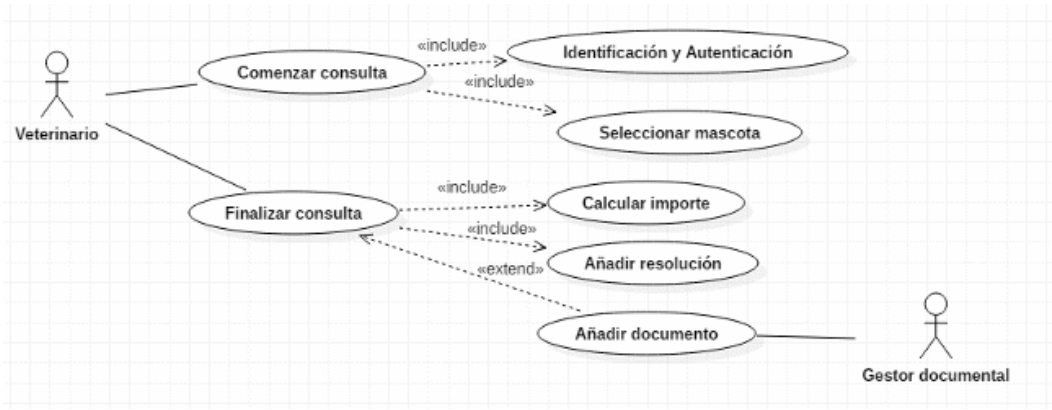

FIGURA 10: DIAGRAMA DE CASOS DE USO[10]

[9] Figura obtenida de: https://diagramasuml.com/wp-content/uploads/2018/09/acr8.png
[10] Figura obtenida de: https://diagramasuml.com/wp-content/uploads/2018/08/Diagrama-de-casos-de-uso_html_m79d70c95.gif

- **Diagrama de máquina de estados:** Diagramas utilizados para significar el estado en el que se encuentran los elementos de un sistema, de modo que se pueda analizar su comportamiento en cualquier situación.

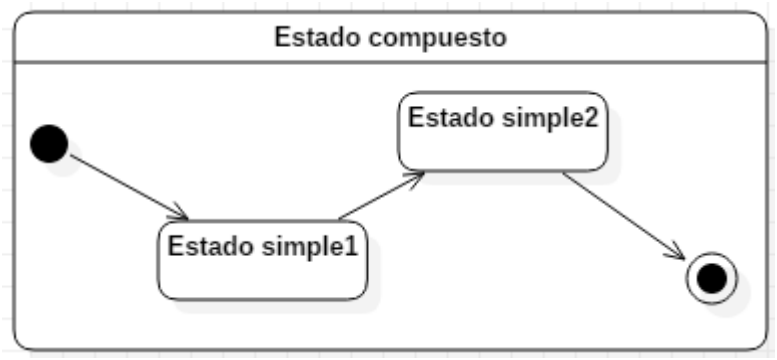

FIGURA 11: DIAGRAMA DE MÁQUINA DE ESTADOS[11]

- **Diagrama de interacción:** A su vez se dividen en otros tantos tipos de diagrama:

    - **Diagrama de secuencia:** Ponen el foco tanto en las relaciones entre los objetos de un sistema, como el orden en que estas se producen. No es objeto de este diagrama mostrar todas las casuísticas posibles, sino centrarse en casos particulares cuya secuencia de ejecución puede resultar interesante analizar. Es algo complejo de entender, dado que los procesos son representados en el eje Y, mientras que las relaciones entre ellos en el tiempo se representan en el eje X.

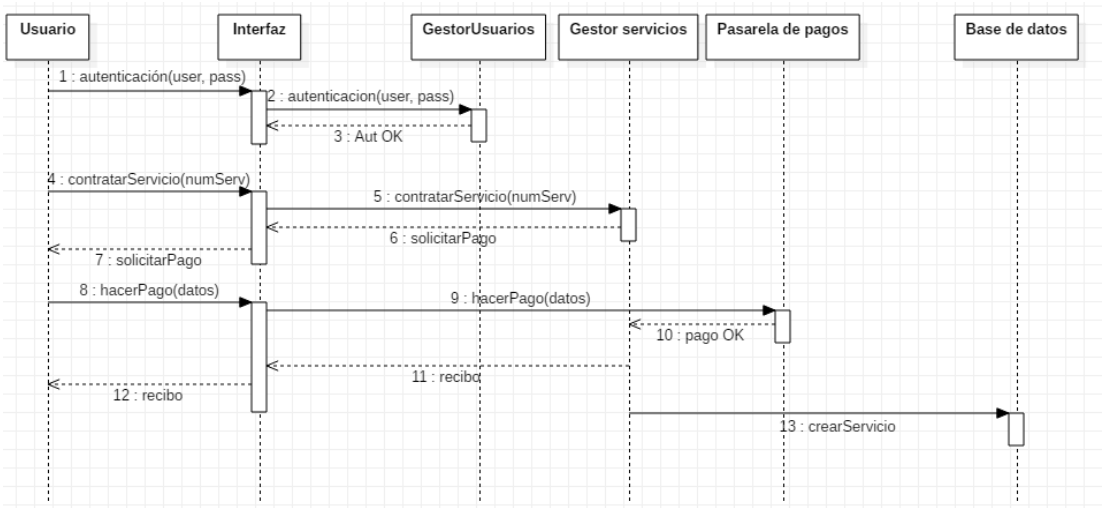

FIGURA 12: DIAGRAMA DE SECUENCIA[12]

[11] Figura obtenida de: https://diagramasuml.com/wp-content/uploads/2018/09/est3.png

- **Diagrama de comunicación:** Su idiosincrasia es la misma que los diagramas de secuencia, sólo que el objetivo es representar las relaciones producidas en el pasado.

- **Diagrama de tiempos:** Como en el caso anterior, se trata de especializaciones de los diagramas de secuencia, en este caso para representar el comportamiento de los objetos de un sistema en una ventana de ejecución determinada.

- **Diagrama global de interacciones:** Se usa la misma notación que los diagramas de actividad, ya que son una especialización de este. Están orientados a mostrar distintos diagramas de interacción englobados unos dentro de otros, así como el orden en que van a suceder.

4. TENDENCIAS DE PATRONES EN LENGUAJES DE PROGRAMACIÓN

Las perspectivas de futuro de los patrones en lenguajes de programación se discuten anualmente en el Pattern Languages Of Programs Conference [18]. La edición de este año se celebra en Octubre, por lo que debemos usar como referencia los topics abordados en la edición de 2018.

Uno de los retos a abordar es conquistar el creciente mundo del Real Time, es decir, desarrollar un lenguaje de patrones para procesos de innovación en tiempo real que represente el conocimiento implícito y tácito para la innovación, mediante un diagrama de flujo dinámico que represente un modelo orientado a la innovación. La combinación de un conocimiento explícito y basado en la experiencia (diagrama de flujo) y un conocimiento implícito/tácito (patrones) puede aplicarse como una herramienta innovadora de planificación y evaluación de la innovación intra-empresarial dentro de las empresas y de las redes empresariales, tanto en la empresa como en la sociedad [19].

Además, los patrones deben adecuarse a la migración progresiva a la nube de la gran mayoría de los negocios de software. Aunque el cloud computing no es un tema nuevo, el diseño de software para la nube todavía requiere que los ingenieros realicen una inversión para ser competentes al trabajar con ella. Por tanto, este será uno de los campos fundamentales de estudio en el futuro: ayudar a los desarrolladores a validar o diseñar su software en la nube [20]. Pero si las empresas con sistemas más antiguos y precloud simplemente cambian a la nube obtienen un beneficio mínimo. Maximizar las ventajas de la infraestructura de la nube requiere un rediseño significativo tanto de los sistemas organizativos como de la cultura [21].

[12] Figura obtenida de: https://diagramasuml.com/wp-content/uploads/2018/08/seq8.png

Esa nube, a la vez debe ser segura, transparente y ética, en un mundo en el que discernir todos estos conceptos es complejo. Los patrones de software también pueden ayudar en este campo. Por ejemplo, es bien sabida la presencia de patrones en tendencias tan crecientes como el blockchain, de modo que es fundamental que los patrones ayuden en la ética de la seguridad de estas tecnologías [22].

Por otro lado, a pesar de que el Big Data está adquiriendo cada vez más importancia en diferentes ámbitos, pocos estudios han investigado la calidad de dichos sistemas desde una perspectiva de la arquitectura de software. El Big Data ha ganado atención principalmente por sus características inherentes de velocidad, veracidad, volumen, valor y variedad, lo que requiere el cumplimiento de varios requisitos como calidad, escalabilidad, rendimiento, seguridad, integridad, flexibilidad, interoperabilidad y objetivos de negocio. Para dar solución a todos estos requisitos, los patrones arquitectónicos son soluciones más que probadas, ya que definen un conjunto de propiedades a alto nivel, un vocabulario para describirlas, y suposiciones y limitaciones sobre cómo deben ser empleadas correctamente. Este campo de estudio debe avanzar al mismo ritmo que lo hace el Big Data [23].

También se han de buscar soluciones orientadas al auge de los asistentes de voz. Apoyar a los usuarios en sus actividades diarias, haciendo sus vidas más cómodas, ha sido durante mucho tiempo un objetivo para el desarrollo de sistemas orientados al consumidor. Sin embargo, con el surgimiento de los asistentes personales inteligentes, se ha alcanzado un nuevo hito en el camino hacia este objetivo. Estos sistemas ayudan a sus propietarios proporcionando información y servicios personalizados y dependientes del contexto. Las implementaciones actuales van desde agentes conversacionales, como Siri, Cortana o Google Assistant, pasando por los chatbots, que se basan principalmente en texto, hasta los asistentes cognitivos, que ayudan en función del estado cognitivo o emocional actual del usuario. Sin embargo, aunque tanto la investigación como la práctica avanzan a buen ritmo, todavía no se han investigado los elementos de diseño recurrentes de los asistentes. Por lo tanto, será necesario proponer un lenguaje de patrones para que los asistentes personales inteligentes guíen los futuros esfuerzos empíricos y de diseño [24].

Esto no son más que proyecciones deseables, pero es más que probable que muchas de ellas se conviertan en realidad a corto-medio plazo, básicamente por la necesidad de estas arquitecturas de adaptarse rápidamente a un mundo cada vez más cambiante.

5. CONCLUSIONES

Como hemos podido ver a lo largo de este trabajo, la fortaleza de ambos conceptos es su principal ventaja. En el caso del lenguaje UML, directamente se ha convertido en el estándar dentro del desarrollo de software, utilizado por prácticamente todos los actores implicados en un proyecto, de manera vertical, desde más alto a más bajo nivel.

Este punto, obviamente, facilita enormemente la comunicación entre todas las partes implicadas, sobre todo con aquellos que no pertenecen a los equipos puramente técnicos. No obstante, no hay que llevarse a engaño, no todo el mundo sabe escribir e interpretar diagramas UML, ni siquiera los profesionales más técnicos, quizás porque muchos de ellos no lo sienten como un lenguaje de código, sino como un lenguaje visual que facilita la comunicación, más orientado a los ingenieros de software. Y esto es un error, porque todas las capas de la organización deberían ser capaces de analizarlos para evitar errores en la interpretación.

Otro de los problemas fundamentales del UML es que en muchos casos no se respeta el estándar, lo que dificulta también la interpretación. La documentación asociada establece claramente cómo dibujar una clase, una agregación o una composición. Si no se respetan estos principios, los errores de interpretación por los distintos actores están garantizados. Quizás la enorme variedad de diagramas distintos no ayude, pero a efectos puramente prácticos, en el entorno laboral basta con dominar los dos principales: el diagrama de clases y el de secuencia. Fundamental dominarlos y entenderlos para evitar malinterpretaciones.

Respecto al Modelo-Vista-Controlador, el hecho de que se use en gran cantidad de organizaciones no es casualidad. Las ventajas que aporta a un desarrollo software son justo aquellas que todo software debería tener. Y es que toda aplicación quiere ser escalable para facilitar el crecimiento; modular para que las partes implicadas puedan trabajar en paralelo, bien durante los diversos evolutivos, y no menos importante, cuando se necesita aplicar un correctivo de urgencia; reusable, con código sencillo de mantener y documentar, lo que además, en entornos web, garantiza rapidez.

Pero también tiene desventajas, pues no todos los desarrollos tienen esas ambiciones. Si un desarrollo es lo suficientemente pequeño, o se trata de un producto estático, no tiene sentido la inversión en tiempo que se necesita para implementar un Modelo-Vista-Controlador, básicamente porque no se va a producir ese retorno de la inversión en el medio-largo plazo.

En cualquier caso, no se puede poner en duda la potencia de ambos paradigmas, y de hecho, siguen siendo el presente y el futuro del desarrollo software.

B. PROYECCIÓN DIDÁCTICA

6. INTRODUCCIÓN.

La Unidad Didáctica está basada en mi experiencia durante el Practicum, ya que los contenidos que se cubren pertenecen a uno de los Ciclos Formativos de Grado Superior en los que desarrollé mi actividad docente, concretamente en el Ciclo Formativo de Grado Superior de Desarrollo de Aplicaciones Web.

Primero contextualizaremos la Unidad Didáctica en el centro que se va a desarrollar, para posteriormente detallar toda la normativa asociada al contenido que se va a impartir, que en este caso está relacionado con el apartado 3 de este trabajo: el lenguaje UML, más concretamente los diagramas de clases.

Se profundizará en la metodología docente utilizada, así como las herramientas de evaluación y recuperación diseñadas como instrumentos de prueba de los conocimientos impartidos.

Por último, se hará referencia al tema transversal tratado en la Unidad Didáctica, así como las actuaciones programáticas para la atención a la diversidad del alumnado.

7. CONTEXTUALIZACIÓN EN EL CENTRO

I.E.S. VIRGEN DEL CARMEN

El IES Virgen del Carmen se encuentra situado en la zona centro de la ciudad de Jaén, con buena comunicación al encontrarse su acceso principal situado en uno de las vías de comunicación más importantes de la capital, tal y como se puede ver en la Figura 13. Esta ubicación hace que en buena medida el alumnado que solicita este Instituto, en cuanto a los niveles básicos (ESO y Bachillerato), no tenga su domicilio familiar en la zona, sino que sea el domicilio de trabajo de alguno de sus progenitores el aportado a la hora de la solicitud de inscripción. Respecto al alumnado de Formación Profesional, el abanico de posibilidades se abre, al existir en el Instituto Ciclos Formativos que tienen un ámbito provincial; no obstante por la ubicación del centro, próximo a la

Estación de Autobuses, no supone este factor una carga añadida para que el alumnado pueda cumplir con su horario de permanencia en el Instituto.

FIGURA 13: LOCALIZACIÓN I.E.S. VIRGEN DEL CARMEN[13]

En general, se trata de alumnos cuyas familias se ocupan, fundamentalmente, en el sector servicios, y que se podrían clasificar dentro de un estrato medio de la población en cuanto a sus niveles de ingresos. Pertenecen, generalmente, a familias estructuradas donde la mayoría de los miembros cohabitan en el núcleo familiar, disponiendo de los espacios básicos para poder realizar sus actividades de estudio y de ocio.

Un grupo de alumnos, menor que el anterior, tienen familias dedicadas al sector primario, pero, en general, las características familiares son similares a las expuestas anteriormente.

Un tercer grupo de alumnos son aquellos que, procedentes de otros países, se integran en el Instituto con un nivel académico inferior a la media de los estudios que están vigentes en el actual sistema educativo y en el centro, y sobre el cual hay que realizar acciones especiales para su integración tanto personal como escolar.

En cualquiera de los casos expuestos, el centro, deberá velar porque la formación, tanto personal como académica de nuestro alumnado, sea lo más integral posible, fomentando el asociacionismo y la participación activa en la vida del mismo organizando actividades que puedan servir tanto para la formación académica como para la formación ciudadana, de manera que la posterior integración de nuestro alumnado, bien para la realización de estudios posteriores, bien en el mundo del

[13] Imagen obtenida de Google Maps.

trabajo, sea lo más satisfactoria posible permitiéndole alcanzar los objetivos que ellos mismos se impongan.

En la generalidad del centro, el nivel de estudios de los padres es medio; en una buena parte de los domicilios existe ordenador y biblioteca que puede ser utilizada por el alumnado. Las aficiones fundamentales del alumnado son el deporte, salir con los amigos, ver la televisión, jugar con videojuegos y aquellas actividades derivadas del uso del ordenador. La participación en las tareas domésticas es mayor entre las alumnas que entre los alumnos, si bien estos últimos también colaboran en buena medida.

La adaptación escolar del alumnado es buena, estando en general satisfechos con los compañeros y con el profesorado y, especialmente los alumnos de los cursos inferiores, se muestran más participativos en las actividades organizadas por el centro, demandando la realización de las mismas. En el centro, además, existe una asociación de padres y madres del alumnado con la que existe una muy buena relación [25].

8. UNIDAD DIDÁCTICA.

Esta Unidad Didáctica, está enmarcada dentro del módulo profesional de Entornos de Desarrollo (código 0487), perteneciente al título de Técnico Superior en Desarrollo de Aplicaciones Web. De acuerdo a la Orden de 16 de junio de 2011 [27], este módulo se imparte en el primer curso, con una programación de tres horas semanales.

Al estar el contenido al final del currículo propuesto por la administración, se han programado las sesiones para ser impartidas durante el tercer trimestre del curso. La duración total de la Unidad Didáctica es de 12h, divididas en 8 sesiones: 4 de 2h y 4 de 1h. Las sesiones de 1h están contempladas con una duración de 55 minutos, mientras que las de 2h tienen una duración de 110 minutos.

Todos los materiales utilizados durante las sesiones se centralizarán en la plataforma Moodle, que además servirá como repositorio de entrega de prácticas. Cada alumno dispondrá de un usuario y contraseña con los que acceder a la plataforma. Para el caso de la entrega de prácticas, la plataforma dispone de un 'periodo de gracia' para la entrega de prácticas, que servirá para conocer cuándo un alumno ha entregado la práctica fuera de tiempo, y así poder evaluarlo en consecuencia.

8.1. NORMATIVA

La Unidad Didáctica está contextualizada dentro del currículo correspondiente al título de Técnico Superior en Desarrollo de Aplicaciones Web, que se rige por la siguiente normativa:

- Real Decreto 686/2010, de 20 de mayo, por el que se establece el título de Técnico Superior en Desarrollo de Aplicaciones Web y se fijan sus enseñanzas mínimas [26].
- ORDEN de 16 de junio de 2011, por la que se desarrolla el currículo correspondiente al título de Técnico Superior en Desarrollo de Aplicaciones Web [27].
- Orden EDU/2887/2010, de 2 de noviembre, por la que se establece el currículo del ciclo formativo de Grado Superior correspondiente al título de Técnico Superior en Desarrollo de Aplicaciones Web [28].

8.2. ELEMENTOS CURRICULARES.

8.2.1. COMPETENCIA GENERAL.

Según el Real Decreto 686/2010 [26], la competencia general del título en el que está enmarcada esta Unidad Didáctica consiste en desarrollar, implantar, y mantener aplicaciones web, con independencia del modelo empleado y utilizando tecnologías específicas, garantizando el acceso a los datos de forma segura y cumpliendo los criterios de accesibilidad, usabilidad y calidad exigidas en los estándares establecidos.

8.2.2. OBJETIVOS GENERALES.

Según la Orden de 16 de junio de 2011 [27], la formación del módulo en el que está contenida esta Unidad Didáctica contribuye a alcanzar los objetivos generales de este ciclo formativo que se relacionan a continuación:

> d) Ajustar parámetros analizando la configuración para gestionar servidores de aplicaciones.
>
> e) Interpretar el diseño lógico, verificando los parámetros establecidos para gestionar bases de datos.
>
> h) Generar componentes de acceso a datos, cumpliendo las especificaciones, para integrar contenidos en la lógica de una aplicación Web.
>
> i) Utilizar lenguajes de marcas y estándares Web, asumiendo el manual de estilo, para desarrollar interfaces en aplicaciones Web.

j) Emplear herramientas y lenguajes específicos, siguiendo las especificaciones, para desarrollar componentes multimedia.

8.2.3. COMPETENCIAS PROFESIONALES, PERSONALES Y SOCIALES.

Según la Orden de 16 de junio de 2011 [27], la formación del módulo en el que está contenida esta Unidad Didáctica contribuye a alcanzar las competencias profesionales, personales y sociales de este título que se relacionan a continuación:

d) Gestionar bases de datos, interpretando su diseño lógico y verificando integridad, consistencia, seguridad y accesibilidad de los datos.

f) Integrar contenidos en la lógica de una aplicación Web, desarrollando componentes de acceso a datos adecuados a las especificaciones.

h) Desarrollar componentes multimedia para su integración en aplicaciones Web, empleando herramientas específicas y siguiendo las especificaciones establecidas.

i) Integrar componentes multimedia en el interface de una aplicación Web, realizando el análisis de interactividad, accesibilidad y usabilidad de la aplicación.

j) Desarrollar e integrar componentes software en el entorno del servidor Web, empleando herramientas y lenguajes específicos, para cumplir las especificaciones de la aplicación.

8.2.4. LÍNEAS DE ACTUACIÓN.

Según la Orden de 16 de junio de 2011 [27], las líneas de actuación en el proceso de enseñanza-aprendizaje que permiten alcanzar los objetivos del módulo en el que está contenida esta Unidad Didáctica versarán sobre:

- La interpretación de documentación técnica.
- La instalación, configuración y personalización de diversos entornos de desarrollo.
- La utilización de distintos entornos de desarrollo para la edición y prueba de aplicaciones.
- La utilización de herramientas de depuración, optimización y documentación de aplicaciones.
- La generación de diagramas técnicos.
- La elaboración de documentación interna de la aplicación.

8.2.5. Orientaciones pedagógicas.

Según la Orden de 16 de junio de 2011 [27], el módulo profesional en el que está contenida esta Unidad Didáctica contiene parte de la formación necesaria para desempeñar la función de desarrollador de aplicaciones.

La función de desarrollador de aplicaciones incluye aspectos como:

- La utilización de las herramientas software disponibles.
- La elaboración de documentación interna y técnica de la aplicación.
- La elaboración y ejecución de pruebas.
- La optimización de código.

8.2.6. Contenidos conceptuales.

Según la Orden de 16 de junio de 2011 [27], el contenido básico que se contextualiza en esta Unidad Didáctica se trata de:

Elaboración de diagramas de clases:

- Notación de los diagramas de clases.
    - Clases. Atributos, métodos y visibilidad.
    - Objetos. Instanciación.
    - Relaciones. Herencia, composición, agregación, asociación y uso.
- Herramientas para la elaboración de diagramas de clases. Instalación.
- Generación de código a partir de diagramas de clases.
- Generación de diagramas de clases a partir de código.

8.2.7. Objetivos didácticos.

- Ser capaz de identificar los conceptos básicos de la programación orientada a objetos.
- Instalar el módulo del entorno integrado de desarrollo que permite la utilización de diagramas de clases.
- Conocer las herramientas para la elaboración de diagramas de clases.
- Saber interpretar el significado de diagramas de clases.
- Trazar diagramas de clases a partir de las especificaciones de las mismas.
- Generar código a partir de un diagrama de clases.
- Generar un diagrama de clases mediante ingeniería inversa.

8.2.8. Resultados de aprendizaje.

Según la Orden de 16 de junio de 2011 [27], el resultado de aprendizaje que aplica a esta Unidad Didáctica es:

- 5. Genera diagramas de clases valorando su importancia en el desarrollo de aplicaciones y empleando las herramientas disponibles en el entorno.

8.2.9. Criterios de evaluación.

Según la Orden de 16 de junio de 2011 [27], los criterios de evaluación que aplican a esta Unidad Didáctica son:

a) Se han identificado los conceptos básicos de la programación orientada a objetos.

b) Se ha instalado el módulo del entorno integrado de desarrollo que permite la utilización de diagramas de clases.

c) Se han identificado las herramientas para la elaboración de diagramas de clases.

d) Se ha interpretado el significado de diagramas de clases.

e) Se han trazado diagramas de clases a partir de las especificaciones de las mismas.

f) Se ha generado código a partir de un diagrama de clases.

g) Se ha generado un diagrama de clases mediante ingeniería inversa.

8.2.10. Temas transversales.

El tema transversal elegido está enmarcado dentro de las actividades programadas por el Día Europeo de las PYMEs, que tiene lugar el 12 de Mayo.

Así, está programada una charla, en la que un responsable de proyecto de una empresa con la que se tenga convenio por el carácter dual de este u otros ciclos, dé una pequeña charla sobre las aplicaciones prácticas en el mundo laboral de todos los conceptos adquiridos hasta ahora, tanto a nivel teórico como práctico. Además, se les trasladará la importancia de las PYMEs para la economía de un país, la importancia de favorecer el comercio local, y cómo una pequeña empresa puede convertirse en un caso de éxito gestionando bien sus recursos humanos.

9. METODOLOGÍA

9.1. ORGANIZACIÓN DEL AULA.

El aula tiene forma rectangular, estando situada la entrada en una de las esquinas laterales, de modo que de frente a la entrada se encuentra la mesa del profesor, y frente a esta 5 líneas de mesas, con un pasillo en medio, teniendo en cada hilera 6 PCs, lo que hacen un total de 30 PCs, más el del docente. Al fondo se encuentra una hilera de percheros y el RAC de comunicaciones, mientras que el lateral frente a la puerta tiene una hilera de ventanales, tal y como se indica en la figura 14.

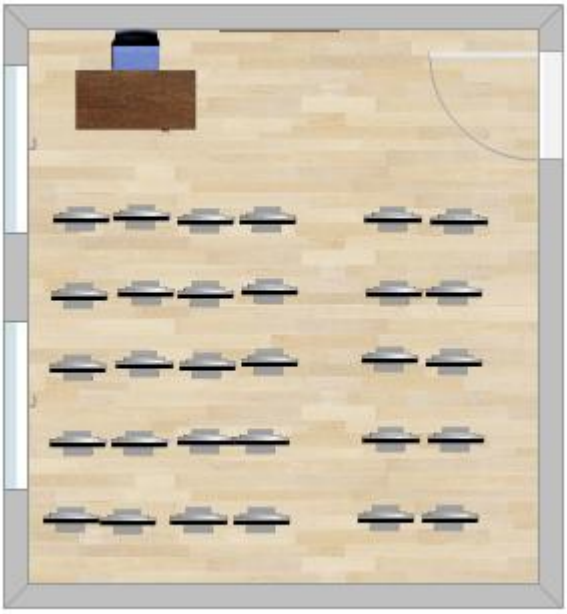

FIGURA 14: AULA PROPUESTA[14]

A la espalda de la mesa del profesor, se dispone de una pequeña pizarra, y del panel para el proyector, situado este último colgado del techo. Cada alumno dispone de un PC exclusivamente para él, siempre el mismo y para todas las clases. En cada PC hay instalado Windows 7 y Ubuntu 18.04, siendo el arranque gestionado por Grub.

Para el desarrollo normal de las clases se usa la plataforma Moodle, de modo que, por un lado, todos los ordenadores tienen conexión a internet para el correcto desarrollo de la sesión, y por otro lado, el uso de la plataforma facilita la gestión de la documentación para el profesor, que no tiene que estar usando pendrives o discos booteables diariamente.

[14] Figura generada por el autor.

9.2. TIPOS DE AGRUPAMIENTO.

Durante las sesiones, se trabajarán los contenidos desde dos tipos de agrupamiento:

- Individual: Durante el desarrollo de contenidos expositivos y sobre todo en las pruebas de carácter práctico, ya que por un lado se pretende fomentar la autonomía del alumno, asegurándonos de que aprenden a utilizar la herramienta ArgoUML por sí mismos; por otro, algunas de las pruebas son de corte evaluatorio y están contempladas para ser puntuadas individualmente.

- Grupal: En la generalidad de las sesiones más teóricas, así como las clases de repaso y resumen, tanto de contenidos como de la prueba con Kahoot. También se hará uso de este tipo de agrupamiento para la charla programada para el Día Europeo de las PYMEs.

9.3. MATERIALES Y RECURSOS DIDÁCTICOS.

De conformidad con lo previsto en el artículo 11.6 del Real Decreto 686/2010, de 20 de mayo [26], los espacios y equipamientos mínimos necesarios para el desarrollo de las enseñanzas de este ciclo formativo son:

Aula polivalente: Aula de entre 40 y 60 m2, que dispone del siguiente equipamiento:

- Equipos audiovisuales.
- Ordenadores instalados en red y con acceso a Internet.
- Cañón de proyección.

Aula de Desarrollo Web: Aula de entre 40 y 60 m2, que dispone del siguiente equipamiento:

- Ordenadores instalados en red y con acceso a Internet.
- Medios audiovisuales: cañón, pantalla de proyección y altavoces.
- Impresora láser y escáner.
- Sistema de alimentación ininterrumpida (SAI).
- Cámara Web.
- Sistemas Operativos.
- Software de control remoto.
- Sistemas Gestores de Bases de Datos. Servidores y clientes.
- Entornos de desarrollo, compiladores e intérpretes, analizadores de código fuente, control de versiones, empaquetadores, generadores de ayudas, entre otros.
- Sistemas de control de versiones.
- Software específico para desarrollo de interfaces Web.

- Software de desarrollo lado cliente.
- Software de desarrollo lado servidor.

Aula de Programación: Aula de entre 40 y 60 m2, que dispone del siguiente equipamiento:

- Ordenadores instalados en red y con acceso a Internet.
- Conexión a la red Internet que permita configurar y redireccionar todos los parámetros y servicios de red.
- Medios de proyección.
- Impresora láser.
- Sistemas Operativos.
- Software de control remoto.
- Servidores de Ficheros, Web, Bases de datos y Aplicaciones.
- Software de creación y edición de máquinas virtuales.
- Herramientas de clonación de equipos.
- Cortafuegos, detectores de intrusos, aplicaciones de Internet, entre otras.
- Sistemas Gestores de Bases de Datos. Servidores y clientes.
- Entornos de desarrollo, compiladores e intérpretes, analizadores de código fuente, empaquetadores, generadores de ayudas, entre otros.
- Dispositivos de interconexión de redes.

9.4. TIPOS DE ACTIVIDADES.

Las sesiones están diseñadas para que la distribución de clases de teoría y práctica sea homogénea, de modo que puedan ser intercaladas para que puedan ser puestos en práctica los conocimientos aprendidos en las clases teóricas. Cuando se finalice una sesión de desarrollo de contenido, se reforzará el contenido al inicio de la sesión posterior.

En esta unidad didáctica se trabajarán los siguientes tipos de actividades:

- **De evaluación de conocimientos previos:** Se realizará una actividad de este tipo al inicio de la unidad didáctica, de modo que apoyándonos en un cuestionario de preguntas V-F, podamos dilucidar los conocimientos de los alumnos tanto a nivel global como individual, de modo que nos ayude a calibrar el resto de sesiones programadas.
- **De desarrollo de contenidos:** Clases de contenido mixto. Por un lado, habrá sesiones de corte teórico, en las que mediante clases magistrales se pretende que el proceso de enseñanza-aprendizaje se produzca de manera exponencial.

Por otro lado, dado el carácter práctico de los contenidos, algunas de las actividades de desarrollo son puramente prácticas.

- **De diagnóstico:** Se realizará una actividad de este tipo, a través de la herramienta Kahoot, con el objetivo de comprobar los conocimientos adquiridos hasta ese momento, para volver a calibrar al grupo a mediados de la programación. Además, esta prueba tendrá un carácter motivador.
- **De desarrollo de práctica:** Estas actividades tendrán un corte totalmente práctico, con el objetivo de aplicar aquellos conocimientos adquiridos durante las sesiones previas. Serán actividades que podrán ser completadas en casa, si durante la sesión programada no se tuviera tiempo suficiente.
- **De resumen:** Estas actividades tienen como objetivo hacer balance de lo visto hasta ahora, con el objetivo de poner en común posibles dudas que hayan podido surgir. Su misión principal es asegurar que se puede seguir avanzando en los contenidos sin dejar a ningún alumno atrás.
- **De refuerzo:** Estas actividades están orientadas a que el alumno afiance conocimientos, de modo que de una manera dinámica nos permitan, por un lado, reforzar conocimientos complejos, y por otro, que lo hagan de una manera natural, sirviéndonos del componente práctico.
- **De ampliación:** Actividades orientadas a profundizar ciertos aspectos del contenido impartido en sesiones previas.
- **De presentación-motivación:** Se trabajará una actividad de este tipo, en la que la dinámica será asistir a una charla, y a su vez generar un debate sobre la utilidad de lo visto hasta ahora, así como su aplicación en el mercado laboral, objetivo último de estos módulos.
- **De evaluación:** Estas actividades tienen como objetivo poner a prueba los conocimientos aprendidos durante las sesiones anteriores. Serán pruebas de corte totalmente práctico.
- **De recuperación:** Esta actividad será análoga a la de evaluación, sirviendo como reválida para aquellos alumnos que no hayan superado la prueba ordinaria.

9.5. TEMPORIZACIÓN DE SESIONES.

Las sesiones en las que se divide esta Unidad Didáctica, así como sus características y programación asociada son:

ACTIVIDAD	DUR.	SES.	AGRUPAM.	RECURSOS
1.1: De evaluación de conocimientos previos. Tipo test de preguntas V-F con Kahoot.	20’	1	Individual	Aula polivalente. Ordenador. Proyector. Dispositivos móviles con conexión a internet.
1.2: De desarrollo de contenidos. Repaso de conceptos básicos asociados a la Programación Orientada a Objetos.	60’	1	Grupal	
1.3: De aplicación práctica de contenidos. Aplicación práctica de los conocimientos previamente repasados.	30’	1	Grupal	
2.1: De aplicación práctica de contenidos. Instalación de la herramienta ArgoUML	25’	2	Individual	Aula técnica. Ordenador. Proyector. Ordenadores instalados en red.
2.2: De desarrollo de contenidos. Introducción a ArgoUML.	30’	2	Individual	
3.1: De desarrollo de contenidos. Inserción de relaciones en ArgoUML.	60’	3	Individual	Aula técnica. Ordenador. Proyector. Ordenadores instalados en red.
3.2: De desarrollo de contenidos. Relaciones de composición en ArgoUML.	50’	3	Individual	
4.1: Desarrollo de práctica. Práctica 1: Elaboración de diagrama de clases sencillo	55’	4	Individual	Aula técnica. Ordenador. Proyector. Ordenadores en red.
5.1: De diagnóstico Kahoot de 15 preguntas para evaluar	15’	5	Individual	Aula técnica. Ordenador.

los conocimientos.				Proyector. Ordenadores instalados en red. Dispositivos móviles con conexión a internet.
5.2: De refuerzo y grupal. Repaso de preguntas del Kahoot.	20’	5	Grupal	
5.3: De desarrollo de contenidos. Relaciones de herencia en ArgoUML.	30’	5	Individual	
5.4: Desarrollo de práctica. Práctica 2: Ampliación de práctica 1.	45’	5	Individual	
6.1: De presentación-motivación. Charla motivadora, enmarcada dentro del Día Europeo de las PYMEs	30’	6	Grupal	Aula técnica. Ordenador. Proyector. Ordenadores instalados en red.
6.2: De ampliación. Generación automática de clases a partir de diagramas, y viceversa.	25’	6	Individual	
7.1: De resumen. Repaso global de todo lo visto hasta ahora.	50’	7	Grupal	Aula técnica. Ordenador. Proyector. Ordenadores instalados en red.
7.2: Desarrollo de práctica final. La práctica final estará dividida en: 1. Elaboración de un diagrama de clases a partir de unas especificaciones. 2. Generación de código automático a partir del diagrama previo. 3. Generación de un diagrama de clases automáticamente a partir de un código dado.	60’	7	Individual	
8.1: De recuperación. Prueba de un corte prácticamente similar al de la prueba ordinaria.	55’	8	Individual	Aula técnica. Ordenador. Proyector. Ordenadores instalados en red.

A continuación, se describirán las sesiones programadas con algo más de detalle.

Sesión 1: Duración: 120 minutos.

Actividad 1.1: De evaluación de conocimientos previos. Duración: 20 minutos. Agrupamiento: Individual.

Los primeros 20 minutos se emplearán en un tipo test de 20 preguntas V-F con el que se pretende obtener una rápida evaluación del nivel general y particular del alumnado sobre la Programación Orientada a Objetos. Se les indicará que la calificación obtenida en esta prueba no es relevante para la nota final del módulo, y de hecho no se les comunicará dicha nota.

Las preguntas serán del tipo:

- Una clase se compone de 3 elementos fundamentales: nombre, atributos y métodos (V/F).
- Las clases se instancia a partir de la palabra 'new' (V/F).
- 'Setter' y 'getter' son tipos de clases (V/F).
- Etc.

Actividad 1.2: De desarrollo de contenidos. Duración 60 minutos. Agrupamiento: Grupal.

Sesión de corte teórico-práctico, en la que se repasarán conceptos básicos de la Programación Orientada a Objetos. No os objeto de esta sesión profundizar en los aspectos de este paradigma, sino hacer un repaso global que sirva de base para en sesiones posteriores poder interpretar de manera más sencilla los diagramas de clases. Se tiene en cuenta que los fundamentos de la Programación Orientada a Objetos se han impartido previamente en el módulo de Programación de este mismo Ciclo Formativo, por lo que se les presupone una base.

En este sentido, durante esta sesión expositiva se repasará el concepto de clase, método, atributo, etc., así como de las relaciones entre ellos. Se hará a partir de unas transparencias que posteriormente los alumnos podrán consultar en la plataforma Moodle. No obstante, dichos conceptos se irán repasando en clases posteriores, a medida que se vayan trabajando los diagramas de clases.

Actividad 1.3: De aplicación práctica de contenidos. Duración: 30 minutos. Agrupamiento: Grupal.

Los últimos 30 minutos se emplearán en hacer una introducción sobre la aplicación práctica de los conocimientos previamente repasados. Así, se les irán proponiendo

diversas clases (Persona, Edificio, Animal, etc), y se irán discutiendo posibles atributos asociados a dichas clases (nombre, edad, peso, altura, localización, etc.), así como posibles métodos que estas clase pudieran contener (esMayordeEdad(), devuelveSexo(), obtenerCoordenadas(), etc.). La idea es discutir estos aspectos desde una perspectiva puramente teórica, a modo de repaso.

Sesión 2: Duración: 60 minutos.

Actividad 2.1: De aplicación práctica de contenidos. Duración: 25 minutos. Agrupamiento: Individual.

Los primeros 25 minutos de la segunda sesión los emplearemos en instalar ArgoUML, una herramienta especialmente pensada para crear diagramas de clase. Es de código libre y se puedes descargar desde su página oficial. Está basada en Java, por lo que es imprescindible tener la máquina virtual de Java instalada (JDK). Es posible que durante la instalación, algunos alumnos tengan diversas incidencias, sobre todo con la JDK, que trataremos de resolver.

Los alumnos que no tengan incidencias durante la instalación de la herramienta, podrán empezar a probar sus funcionalidades, mientras sus compañeros las solucionan.

Actividad 2.2: De desarrollo de contenidos. Duración: 30 minutos. Agrupamiento: Individual.

En la sesión previa, definimos diversas clases, con una serie de métodos y atributos asociados. Por tanto, usaremos estos ejemplos para introducir la herramienta ArgoUML para el diseño de diagramas de clase. De esta forma, crearemos las clases, de momento desconectadas entre sí, mientras vamos repasando las distintas opciones que la herramienta nos da para representar las clases:

- Notación de la visibilidad de los atributos o métodos.
- Tipos de métodos.
- Etc.

Sesión 3: Duración: 120 minutos.

Actividad 3.1: De desarrollo de contenidos. Duración: 60 minutos. Agrupamiento: Individual.

La primera parte de la sesión 3 la emplearemos en seguir profundizando en la forma de representar los diagramas de clases. Así, como en la última sesión vimos cómo

representar una clase, dedicaremos esta primera parte de la sesión en ir repasando las formas de representar las relaciones entre clases.

De esta forma, en el proyector, usaremos la herramienta ArgoUML para crear las clases Empleado y Empresa, con una serie de atributos y métodos que de una forma dinámica vayan surgiendo durante su creación. Una vez definidas las clases, representaremos la asociación entre ellas, de modo que hagamos repaso de cada una de las opciones disponibles:

- Multiplicidad.
- Nombre de la asociación.

Ellos deben ir desarrollando el diagrama de clases de ejemplo a la vez que se va realizando en el proyector. Una vez finalizada la construcción, la salida debe ser similar a la de la Figura 15.

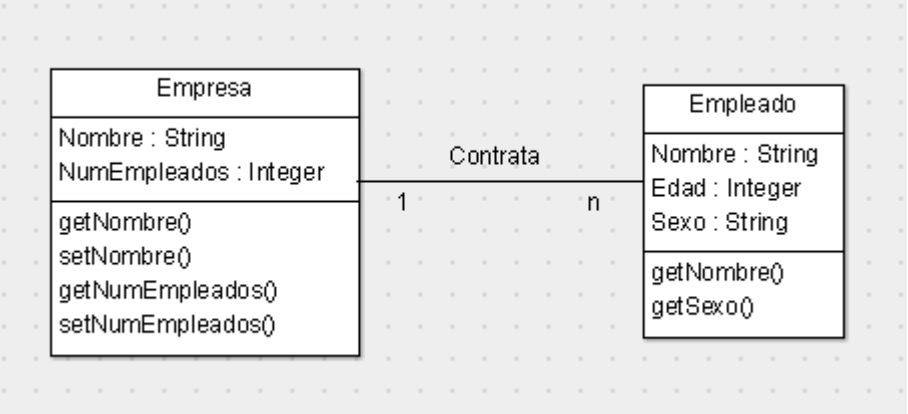

FIGURA 15: DIAGRAMA DE CLASES ARGOUML[15]

Actividad 3.2: De desarrollo de contenidos. Duración: 50 minutos. Agrupamiento: Individual.

La dinámica de la segunda parte de la sesión será similar, con la única diferencia que los alumnos ya tendrán cierta soltura en el uso de la herramienta. Así, iremos construyendo un diagrama similar a la anterior, pero esta vez con el objeto de representar una composición.

Así, deberán ampliar el diagrama anterior, esta vez añadiendo una composición sobre la clase 'Empresa' o 'Empleado'. Esta composición es libre, es decir, cada alumno podrá representar la clase que compone a 'Empresa' o 'Empleado' de la manera que crea más oportuno, de este modo a la misma vez que aprenden a usar la herramienta, comprobamos si han entendido el concepto. Esta vez, a diferencia de la actividad

[15] Figura generada por el autor.

anterior, no se irá construyendo el diagrama en el proyector, sino que cada alumno lo irá construyendo individualmente e iremos resolviendo las posibles dudas que puedan surgir. La idea de hacerlo así es para que el alumno sea más autónomo a la hora de buscar las distintas opciones que proporciona la herramienta.

El contenido de esta sesión al completo debe haber sido visto ya por el alumno en el módulo de Programación, por lo que estos conceptos, además de ser necesarios para la construcción del diagrama de clases, servirán de repaso de lo visto.

Sesión 4: Duración: 60 minutos.

Actividad 4.1: Desarrollo de práctica. Duración: 55 minutos. Agrupamiento: Individual.

Una vez trabajados los ejemplos de la sesión anterior, se les darán unas especificaciones para que puedan elaborar por su cuenta un diagrama de clases sencillo, mediante la herramienta ArgoUML. Estas especificaciones estarán a disposición del alumnado en la plataforma Moodle.

Esta sesión supondrá la entrega de la Práctica 1. No obstante, si no han conseguido completarla a tiempo, podrán hacerlo desde casa antes del comienzo de la sesión siguiente. Para ello, se les habilitará una entrega a través de la plataforma Moodle.

Las especificaciones que se les propondrán serán la construcción de un diagrama que contenga las clases: Universidad, Profesor, Departamento y Alumno. Cada clase deberá contener 2 atributos y los métodos getter/setter. De igual forma, las relaciones deberán contener el nombre y la multiplicidad.

Sesión 5: Duración: 120 minutos.

Actividad 5.1: De diagnóstico. Duración: 15 minutos. Agrupamiento: Individual.

Durante los primeros 15 minutos de la sesión, se realizará un Kahoot de 15 preguntas para evaluar si los conocimientos impartidos hasta ahora se están asentando. Además, previamente se les comunicará que los 3 mejores alumnos en el Kahoot, obtendrán 0’75, 0’5 y 0’25 extra en la nota final, respectivamente. El formato del Kahoot será preguntas tipo test con 4 posibles respuestas, siendo únicamente una de ellas válida. Las preguntas del Kahoot tendrán un corte teórico-práctico.

Actividad 5.2: De refuerzo y grupal. Duración: 20 minutos. Agrupamiento: Grupal.

Tras la finalización del Kahoot, y analizando al momento los resultados obtenidos, se irán repasando una a una todas las preguntas, de modo que se puedan ir solventando las diversas dudas que hayan ido surgiendo durante la realización de la actividad. Se

interpelará a los alumnos que hayan fallado ciertas preguntas clave para profundizar en los errores y tratar de afianzar esos conocimientos. El objetivo de esta actividad de refuerzo es asegurarnos de que podemos avanzar en los contenidos sin que nadie se quede atrás.

Actividad 5.3: De desarrollo de contenidos. Duración: 30 minutos. Agrupamiento: Individual.

En esta actividad incorporaremos el concepto de 'herencia' a los diagramas de clases. Es un concepto que ya deben conocer del módulo de Programación, por lo que lo que haremos será realizar un diagrama de ejemplo en el proyector, con la idea de que vean aplicado de una forma visual el concepto.

A estas alturas de la Unidad Didáctica, deben tener ya soltura con la herramienta ArgoUML, por lo que cada vez será necesario menos tiempo para explicar el uso de la herramienta, y así poder centrarnos en la potencia de la herramienta en sí.

Para la explicación de la herencia, realizaremos un diagrama de clases de ejemplo, en el que estará incorporada la clase Animal, así como tres clases de ejemplo que heredan de ella, como pueden ser Pez, Perro y Gato. El diagrama de ejemplo sería similar a la de la Figura 16.

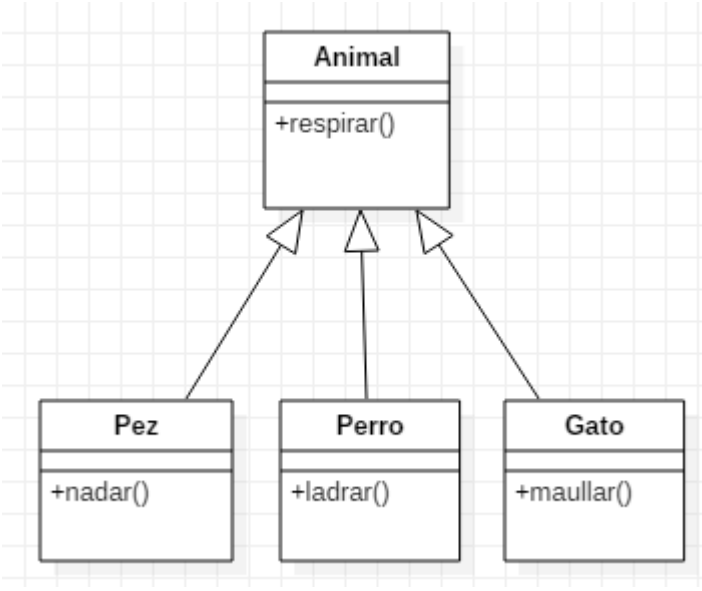

FIGURA 16: EJEMPLO DIAGRAMA DE CLASES: HERENCIA[16]

[16] Figura obtenida de: https://diagramasuml.com/wp-content/uploads/2018/08/clases6.png

Actividad 5.4: Desarrollo de práctica. Duración: 45 minutos. Agrupamiento: Individual.

Se explicará en qué consiste la segunda práctica, en la que se aplicarán los conceptos aprendidos en las sesiones previas. Así, partiendo del diagrama realizado en la primera práctica, se ampliará, de modo que se incorpore el concepto de agregación recién visto en la actividad previa. Estas especificaciones estarán a disposición del alumnado en la plataforma Moodle.

Así, deberán modificar el diagrama para añadir la clase Persona, y por tanto, modificar en consecuencia las clases que ya existan y que hereden de esta clase nueva, que en el caso que nos ocupa, serían las clases 'Profesor' y 'Alumno'.

Si no han conseguido completar la práctica en el tiempo previsto, podrán hacerlo desde casa y entregarla antes del comienzo de la sesión siguiente. Para ello, se les habilitará una entrega a través de la plataforma Moodle.

Sesión 6: Duración: 60 minutos.

Actividad 6.1: De presentación-motivación. Duración: 30 minutos. Agrupamiento: Grupal.

Una vez completada una parte significativa de los contenidos previstos en las sesiones que completan el módulo, se programará una exposición, enmarcada dentro del Día Europeo de las PYMEs, establecido el 12 de Mayo, en la que un responsable de proyecto de una empresa con la que se tenga convenio por el carácter dual de este u otros ciclos, dé una pequeña charla sobre las aplicaciones prácticas en el mundo laboral de todos los conceptos adquiridos hasta ahora, tanto a nivel teórico como práctico. El objetivo de esta actividad es hacerles llegar a la conclusión de la potencia de los diagramas de clases, sobre todo al inicio de un proyecto, donde la idea que se representa en el diagrama debe ser vendida. Además, se les trasladará la importancia de las PYMEs para la economía de un país, la importancia de favorecer el comercio local, y cómo una pequeña empresa puede convertirse en un caso de éxito gestionando bien sus recursos humanos.

La asistencia a esta charla tendrá carácter obligatorio, de modo que sólo se permitirán asistencias justificadas. La penalización por no asistir se detallará en el apartado correspondiente de esta Unidad Didáctica.

Esta tarea servirá de introducción para dos aplicaciones prácticas muy útiles asociadas a los diagramas de clases: la generación de código a partir de ellos y la generación automática de diagramas a partir del código.

Actividad 6.2: De ampliación. Duración 25 minutos. Agrupamiento: Individual.

En la línea de la charla recién finalizada, usando ArgoUML, la herramienta con la que se ha trabajado en el resto de sesiones, dos funcionalidades de la aplicación enormemente potentes y sencillas. Para ello haremos uso del proyector.

Así, por un lado se les enseñará cómo generar automáticamente el código de las clases contenidas en un diagrama de clases. En ArgoUML la opción es bien sencilla, y realmente no merece mucha más explicación. Para ello, deberán abrir cualquiera de los diagramas trabajados hasta ahora, y a través del menú 'Generar', pinchar en 'Generar todas las clases', tal y como se indica en la Figura 17.

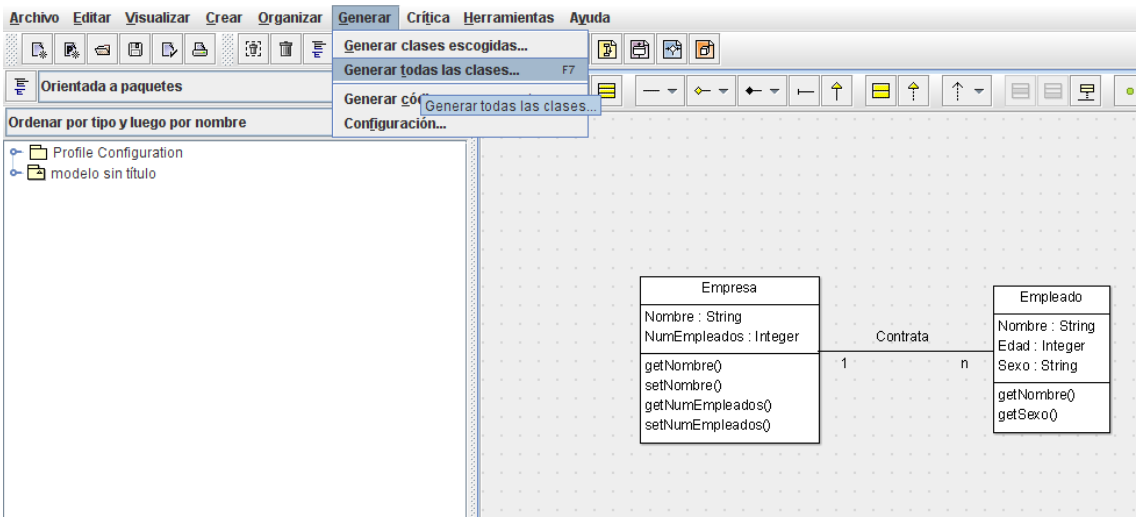

FIGURA 17: EJEMPLO CÓDIGO AUTOMÁTICO ARGOUML[17]

Una vez dentro, deberemos indicar el lenguaje de programación en el que queremos generar el código, así como la carpeta de destino.

Por otro lado, se enseñará también la opción inversa: generar el diagrama de clases a partir de un código ya disponible. Esta opción es sumamente sencilla, simplemente tendrán que entrar al menú 'Archivo' y pinchar en 'Importar desde código', tal y como se indica en la Figura 18. Para probar su funcionamiento, importarán las mismas clases que han generado automáticamente.

[17] Figura generada por el autor.

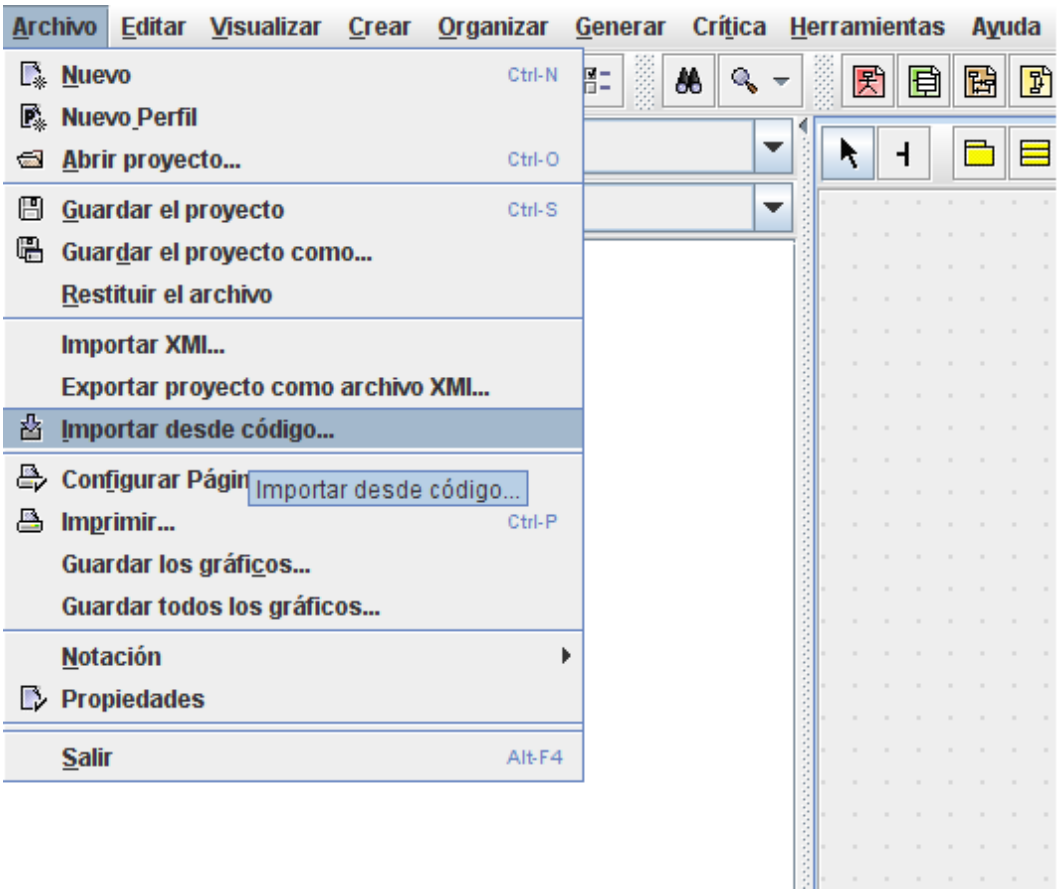

FIGURA 18: EJEMPLO DIAGRAMA AUTOMÁTICO ARGOUML[18]

El objetivo de esta explicación es orientar al alumno de cara a la última sesión práctica. Aunque son conceptos relativamente complejos, no es necesaria una sesión de más duración pues las opciones dentro de la herramienta son bastante intuitivas.

Sesión 7: Duración: 120 minutos.

Actividad 7.1: De resumen. Duración: 50 minutos. Agrupamiento: Grupal.

Emplearemos la primera parte de esta última sesión en hacer un repaso global de todo lo visto hasta ahora, de modo que puedan acceder a la última práctica con garantías. Se focalizará en los últimos contenidos trabajados, siempre desde un punto de vista puramente práctico. Se interpelará a los alumnos sobre posibles dudas.

Actividad 7.2: Desarrollo de práctica final. Duración: 60 minutos. Agrupamiento: Individual.

La práctica final estará dividida en varias partes:

1. Por un lado, se les darán unas especificaciones, entre las que se encontrarán una serie de clases que tendrán que dibujar en un diagrama. Estas clases

[18] Figura generada por el autor.

estarán relacionadas con las necesarias para establecer un sistema de reservas hotelero sencillo:

- Hotel.
- Habitación.
- Reserva.
- Empleado.
- Cliente.
- TipoHabitación.

Esta tarea deberá ser completada en ArgoUML. Una vez completado el diagrama, deberán guardarlo como imagen PNG y subirlo a la plataforma Moodle, en una entrega que estará habilitada expresamente.

2. Después, deberán generar automáticamente el código asociado a cada una de las clases creadas, el cual deberá ser entregado también en la plataforma Moodle.
3. La última fase de la prueba práctica consistirá en la generación de un diagrama de clases automáticamente a partir de un código dado. Pero este código contendrá ciertos errores que deberán solucionar para que la herramienta sea capaz de generar el diagrama.

Para ambas prácticas, podrán disponer en todo momento de los apuntes disponibles en la plataforma Moodle, utilizados en el resto de sesiones, así como de los propios apuntes que hayan tomado en clase.

Sesión 8: Duración: 60 minutos.

Actividad 8.1: De recuperación. Duración: 55 minutos. Agrupamiento: Individual.

Para los alumnos que hayan suspendido la Unidad Didáctica, tras tener en cuenta todos los apartados correspondientes a la evaluación, dispondrán de una segunda oportunidad, con una prueba de un corte prácticamente idéntico al de la prueba ordinaria.

10. EVALUACIÓN Y RECUPERACIÓN.

10.1. PROCEDIMIENTOS E INSTRUMENTOS.

- Cuaderno del profesor.
- Entrega de prácticas.
- Realización de prueba práctica final.
- Asistencia a charla motivadora.
- Kahoot.
- Actitud.

10.2. CRITERIOS DE CALIFICACIÓN.

La Unidad Didáctica se evaluará a partir de los siguientes ítems:

a) **Entrega de prácticas:** 20% de la nota final.
   Se trata de 2 prácticas de entrega opcional. No es necesario entregar las prácticas para superar la asignatura.

b) **Realización de práctica final:** 70% de la nota final.
   Tanto para evaluar las prácticas, como la prueba final, se usará la rúbrica indicada más abajo. En ella, se partirá con una nota de 4 (mínima para poder superar esta Unidad Didáctica), la cual irá aumentando o disminuyendo en función de la siguiente tabla:

Mal (-1)	Regular (0)	Bien (+1)	Muy Bien (+2)
Las clases están representadas con errores de formato.	Las clases no tienen errores de formato, pero los atributos son incorrectos.	El formato es correcto, los atributos también, pero los métodos implementados no son los adecuados.	Las clases se han definido y dibujado de manera correcta.
El diagrama de clases contiene 3 o más errores en las relaciones.	El diagrama de clases contiene 2 errores en las relaciones.	El diagrama de clases contiene 1 error en las relaciones.	El diagrama de clases dibuja las relaciones sin errores.

Las notaciones y la multiplicidad están mal representadas.	Las notaciones están bien, pero no así la multiplicidad de alguna relación.	Las multiplicidades están bien, pero no así la notación de alguna relación.	Tanto las notaciones como las multiplicidades están representadas correctamente.

La nota obtenida tras evaluar esta rúbrica supondrá el 80% de la nota de esta parte. Además, la prueba final incorpora otras dos entregas, que supondrán un 10% cada una de la nota total de esta parte.

c) **Evaluación de la actitud y asistencia:** 10% de la puntuación total.
Cada alumno/a parte con una nota de 4, que aumentará o disminuirá a partir del siguiente criterio:
   - Falta asistencia -0.25 por cada falta.
   - Retraso: -0.1 punto.
   - Positivos: +1 punto, hasta un máximo de 6.
   - Parte comportamiento: -10 puntos.
   - Falta de respeto a compañeros o profesor: -10 puntos.
   - Mal uso de las herramientas del centro: -10 puntos.

Para calcular la **nota final**: Practicas*2*10% + Práctica Final*70% + Actitud/Asistencia*10%. Además, los 3 primeros en la clasificación tras el kahoot de la actividad 5.1, obtendrán un extra a sumar en la nota, de un 0’75, 0,5 y 0’25 respectivamente.

Para superar la Unidad Didáctica, será necesario obtener una nota superior a 5. La no asistencia a la charla, de carácter obligatoria, de forma no justificada, supondrá automáticamente suspender esta Unidad Didáctica.

10.3. SISTEMAS DE RECUPERACIÓN.

Aquellos alumnos que no hayan obtenido una calificación igual o superior a 5 en la convocatoria ordinaria, podrán:

- Realizar una prueba recuperatoria al final de la Unidad Didáctica y del trimestre.
- Las pruebas serán de corte similar a la práctica final de la convocatoria ordinaria.
- Se calificará como APTO o NO APTO, siendo APTO una nota de 5 y NO APTO una nota de 1.

11. ATENCIÓN A LA DIVERSIDAD

La Ley 17/2007, de 10 de diciembre, de Educación en Andalucía, en su artículo 113 (Principios de equidad) define alumnado con necesidades educativas específicas de apoyo educativo (n.e.a.e) como aquel que presenta necesidades educativas especiales debidas a diferentes grados y tipos de capacidades personales de orden físico, psíquico, cognitivo o sensorial; el que, por proceder de otros países o por cualquier otro motivo, se incorpore de forma tardía al sistema educativo, al alumnado con altas capacidades intelectuales, así como el alumnado que precise de acciones de carácter compensatorio [29].

Respecto al caso que nos ocupa, sobre la atención a la diversidad en los ciclos formativos, la Ley 17/2007, de 10 de diciembre, en el Artículo 69 establece que la Administración educativa establecerá medidas de acceso al currículo, así como, en su caso, adaptaciones y exenciones del mismo, dirigidas al alumnado con discapacidad que lo precise en función de su grado de minusvalía [29].

Contextualizado dentro de nuestra Unidad Didáctica, el alumnado no necesita adaptaciones significativas en el currículo, pero en la primera actividad, se ha previsto realizar una prueba, usando Kahoot, con la que se pretende obtener una visión tanto global como individual de las posibles necesidades del alumnado. La potencia que ofrece Kahoot en los resultados una vez terminada la prueba, nos permitirá adaptar el ritmo de aprendizaje de aquellos alumnos que así lo requieran.

Por otro lado, se ha establecido que los ejercicios prácticos que se realicen en las sesiones intermedias tengan carácter opcional, lo que a efectos prácticos supone que el alumnado que no sea capaz de entregarlos a tiempo no se queda atrás, y además, se refuerza a aquellos alumnos más avanzados.

También se han establecido dos actividades especialmente diseñadas para servir de repaso, a modo de tutoría colectiva. Una de estas sesiones está programada para ser realizada previamente a la práctica final. En este sentido, durante estas sesiones se pretende que la dinámica de clase favorezca la interacción alumno-alumno entre aquellos con más capacidades y aquellos con menos.

En el caso de detectar algún alumno que tenga necesidades educativas significativas, se comunicará con el departamento de orientación, así como con los padres, para que se puedan establecer las medidas necesarias.

12. BIBLIOGRAFÍA

[1] Desarrollo Web. *Qué es MVC*. Recuperado desde https://desarrolloweb.com/articulos/que-es-mvc.html [Fecha de consulta de la página: 07/06/2019]

[2] Openclassrooms. *El patrón Modelo–vista–controlador (MVC)*. Recuperado desde https://openclassrooms.com/en/courses/4309566-descubre-la-arquitectura-mvc/4942546-el-patron-modelo-vista-controlador-mvc [Fecha de consulta de la página: 07/06/2019]

[3] Página personal de Trygve Reenskaug. Recuperado desde http://heim.ifi.uio.no/~trygver/trygve/trygve.html [Fecha de consulta de la página: 04/06/2019]

[4] Krasner, G & Pope, S. (1988). *A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk -80*. Journal Of Object Oriented Programming. Volume 1 Issue 3, Pages 26 – 49.

[5] Burbeck, S. (1987). *Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)*.

[6] Reenskaug, T. & Coplien, J.O. (2009). *The DCI Architecture: A New Vision of Object-Oriented Programming*.

[7] Servicio de Informática. Universidad de Alicante. < Recuperado desde https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html [Fecha de consulta de la página: 04/06/2019]

[8] Marco de Desarrollo de la Junta de Andalucía. *Patrón Modelo Vista Controlador*. Recuperado desde http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/122 [Fecha de consulta de la página: 03/06/2019]

[9] Coding or Not. *MVC (Modelo-Vista-Controlador): ¿qué es y para qué sirve?*. Recuperado desde https://codingornot.com/mvc-modelo-vista-controlador-que-es-y-para-que-sirve [Fecha de consulta de la página: 07/06/2019]

[10] TP (2000). Is HMVC PAC? (letter to the editor). JavaWorld.

[11] Zamudio, L. S. A.; Salgado, R. S. & Fragoso, O. G. D. (2012). *Restructuring Object-Oriented Frameworks to Model-View-Adapter Architecture*. IEEE Latin America Transactions. 10 (4): 2010–2016.

[12] Lamarca, M. J. (2013). *Hipertexto: el nuevo concepto de documento en la cultura de la imagen*. Universidad Complutense de Madrid.

[13] Fowler, M. & Scott, K. (1999). *UML gota a gota*.

[14] Lucidchart. *Qué es el lenguaje unificado de modelado (UML).* Recuperado desde https://www.lucidchart.com/pages/es/que-es-el-lenguaje-unificado-de-modelado-uml [Fecha de consulta de la página: 07/06/2019]

[15] OMG. *About the unified modeling language specification version 2.5.1*. Recuperado desde https://www.omg.org/spec/UML/ [Fecha de consulta de la página: 07/06/2019]

[16] Rumbaugh, J; Jacobson, I. & Booch, G. (1999). *The unified modeling language: Reference manual.* Addison Wesley.

[17] Aprender a programar. *¿Qué es y para qué sirve UML? Versiones de UML (Lenguaje Unificado de Modelado). Tipos de diagramas UML.* Recuperado desde https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=688:ique-es-y-para-que-sirve-uml-versiones-de-uml-lenguaje-unificado-de-modelado-tipos-de-diagramas-uml&catid=46&Itemid=163 [Fecha de consulta de la página: 07/06/2019]

[18] Pattern Languages of Programs Conference. Recuperado desde https://www.hillside.net/plop/2019/ [Fecha de consulta de la página: 13/06/2019]

[19] Sailer, K. *Real Time Innovation - Change the pattern. Change your thinking.* Recuperado desde https://www.sce.de/en/realtimeinnovation.html [Fecha de consulta de la página: 13/06/2019]

[20] Sousa, T.B., Ferreira, H.S. & Correia, F.F.. (2018). *Overview of a Pattern Language for Engineering Software for the Cloud.* HILLSIDE Proc. of Conf. on Pattern Lang. of Prog.

[21] Reznik, P. (2018). *Cloud native transformation pattern language.*

[22] Tedeschi, M. (2018). *Ethical Decision Making Patterns*. College of Technology.

[23] Sena, B.; Garcés, L; Allian, A.P. & Nakagawa, E.Y. (2018). *Investigating the Applicability of Architectural Patterns in Big Data Systems*.

[24] Knote, R., Söllner, M. & Leimeister, J.M. (2018). *Towards a Pattern Language for Smart Personal Assistants*. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog.

[25] Plan de Centro del I.E.S. Virgen del Carmen. Recuperado desde https://www.iesvirgendelcarmen.com/ies/sites/default/files/documentos/PLAN%20DE%20CENTRO.pdf

[26] Real Decreto 686/2010, de 20 de mayo, por el que se establece el título de Técnico Superior en Desarrollo de Aplicaciones Web y se fijan sus enseñanzas mínimas.

[27] ORDEN de 16 de junio de 2011, por la que se desarrolla el currículo correspondiente al título de Técnico Superior en Desarrollo de Aplicaciones Web.

[28] Orden EDU/2887/2010, de 2 de noviembre, por la que se establece el currículo del ciclo formativo de Grado Superior correspondiente al título de Técnico Superior en Desarrollo de Aplicaciones Web.

[29] Ley 17/2007, de 10 de diciembre, de Educación en Andalucía.