**Cuestiones tema 4**

**1)** **En este tema hemos visto una clasificación arquitectónica que agrupa los sistemas software en grandes categorías o estilos, cada uno con características propias. Cada estilo arquitectónico impone un mecanismo de comunicación que incide en la independencia de los componentes y determina la mayor o menor facilidad para extender el sistema o reutilizar componentes.**

**De acuerdo con esto, ordenar los siguientes tipos de sistemas de menor a mayor independencia entre componentes, justificando el orden establecido:**

1. **sistemas de filtros y tuberías (por ejemplo, los construidos con la shell de Unix)**

Muy independiente -> 7

En esta arquitectura los filtros (conectados mediante tuberías) son independientes unos de otros. Son sistemas altamente composicionales y extensibles; y sus componentes son reutilizables.

1. **sistemas basados en programación estructurada (por ejemplo, los programados en C)**

Sistema operativo y el propio programa compilado -> 2

La programación estructurada generalmente son programas especializados para un sistema en concreto, haciendo de esta manera complicado la reutilización de algunos componentes.

1. **sistemas basados en programación orientada a objetos (por ejemplo,los programados en Java)**

Máquinas virtuales -> 6

Estos sistemas permiten la facilidad de portabilidad de los sistemas, alto nivel de abstracción, flexibilidad e interacción. Como se crean clases estos sistemas suelen ser más fáciles de reutilizar.

1. **sistemas cliente/servidor (por ejemplo, un sistema de intercambio de archivos mediante FTP)**

Componentes Independientes -> 5.

La arquitectura cliente/servidor, se basa en una red de comunicación en la cual los clientes están conectados al servidor, es decir, las gestiones que se realizan se centran en el servidor, de ahí su alta dependencia.

1. **sistemas de brokering (por ejemplo, los desarrollados de acuerdo a la arquitectura CORBA)**

Poco independiente -> 1.

De los sistemas vistos anteriormente es el más dependiente, ya que como en la arquitectura CORBA, el sistema necesita la conexión de todos sus componentes, véase la serialización.

1. **sistemas de eventos (por ejemplo, la interfaz gráfica de Windows)**

Componentes independientes -> 4.

Son sucesos provocados por el usuario, estos eventos vendrán determinados dependiendo del lenguaje de programación utilizado, incluso el sistema operativo los cuales ejercerán un control determinado dependiendo de las acciones del usuario.

1. **sistemas de repositorio (por ejemplo, un sistema de gestión empresarial típico, centrado en una base de datos).**

Basado en los datos -> 3.

Sistema basado en el almacenamiento centralizado de datos. No hay comunicación directa entre los componentes y el repositorio actúa de vínculo.

**2)** **Discutir las características arquitectónicas de un sistema de intercambio de archivos como pueden ser eMule o bitTorrent!, indicando el estilo arquitectónico en el que podríamos encuadrarlos, de entre los vistos en este tema, o justificando si sería necesario incluir un nuevo estilo o subestilo dentro de dicha clasificación. Asimismo, se pide comparar ambos sistemas (eMule y bitTorrent!) y discutir las diferencias entre ellos.**

El sistema que usa eMule es basado en la arquitectura de componentes independientes ya que está compuesto de muchos clientes y varios servidores. Sin embargo, no sigue la estructura de cliente-servidor, puesto que, a la hora de la descarga de los datos, los clientes se conectan entre ellos directamente, mientras que en la arquitectura cliente-servidor, siempre se comunican a través del servidor. El servidor en eMule solo sirve para almacenar qué usuario tiene qué ficheros. Por tanto, eMule no sigue la arquitectura cliente-servidor, si no que sigue la arquitectura Peer-to-peer, la cual podríamos considerarla como un subestilo dentro del estilo arquitectónico de componentes independientes.

De igual forma, bitTorrent utiliza la misma arquitectura que eMule. La única diferencia es cómo accede el cliente al servidor, en el caso de eMule, el propio cliente puede acceder directamente al servidor haciendo una búsqueda de los ficheros que se encuentran en ese servidor, sin embargo, en bitTorrent, la búsqueda se realiza mediante el uso de un fichero .torrent o un magnet, que realiza la búsqueda automáticamente.

**3)** **Comparar los siguientes conceptos de UML, identificando sus similitudes y diferencias:**

1. **Clase vs. Actor**

Un actor es una entidad física mientras que la clase es una abstracción lógica. Además, la clase pertenece al sistema, sin embargo, el actor es externo a éste sistema. Se asemejan en que ambos interactúan con el sistema.

1. **Componente vs. Clase**

Ambos tienen nombre y definen unas propiedades y pueden realizar un conjunto de interfaces, participar en relaciones e interacciones, ambos pueden participar en relaciones e interacciones. Las clases representan abstracciones lógicas y los componentes son el empaquetamiento físico de estos elementos lógicos. Por último, los componentes solo pueden ser accesibles mediante interfaces, sin embargo, las clases no necesitan la interfaz para acceder a ellas.

1. **Clase vs. Interfaz**

Las clases e interfaces ambas participan en relaciones, sin embargo, las interfaces no son capaces de interactuar con el sistema por sí solas puesto que son símplemente operaciones sin cuerpo y atributos. Las clases pueden completar la definición de interfaces para que éstas puedan comunicarse con el sistema.

1. **Clase vs. Clase Abstracta**

Ambas contienen una declaración de atributos y operaciones y ambas son abstracciones lógicas, además, ambas pueden participar en relaciones. Sin embargo, las clases abstractas por sí solas no son capaces de interactuar con el sistema puesto que no todas las operaciones están definidas, mientras que las clases, si que son capaces de interactuar con el sistema puesto que si que tienen el cuerpo en todas las operaciones.

1. **Clase Abstracta vs. Interfaz**

Ambas contienen una declaración de atributos y operaciones y ambas son abstracciones lógicas. Además, ambas pueden participar en relaciones y no son capaces de interactuar con el sistema por sí solas. Sin embargo, las clases abstractas implementan parte de sus operaciones mientras que las de las interfaces no están implementadas.

1. **Puerto vs. Interfaz**

Ambos permiten a los componentes comunicarse con el sistema exterior, sin embargo, un puerto es un conjunto de interfaces.

1. **Componente vs. Puerto**

Los Puertos son puntos de interacción entre un componente y el exterior. Un componente encapsula su funcionalidad e interactúa con el exterior del mismo a través de sus puertos, que representan la delegación y dependencia de sus interfaces.

1. **Artefacto vs. Componente**

El componente hace uso del artefacto, puesto que un artefacto puede ser una biblioteca, una tabla de la bbdd, etc. Ambos son representaciones de conceptos, sin embargo, el componente representa la parte lógica, y el artefacto es como “la instancia” en un momento dado de una representación lógica.

1. **Artefacto vs. Clase**

La clase representa una abstracción lógica de parte de un sistema mientras que el artefacto es un fichero físico, como puede ser una biblioteca, en la que contenga más de una clase compilada. Ambos representan parte del sistema, sin embargo, una es la implementación del sistema a nivel físico (Artefacto) mientras que la clase es una representación lógica del mismo.

1. **Nodo vs. Componente**

Los nodos son estructuras físicas, mientras que los componentes, son estructuras lógicas.

**4) Define el “principio de Hollywood” que se usa en los marcos de trabajo, y discute por qué este principio de inversión de control no facilita la composición de marcos de trabajo.**

El principio de Hollywood, o de Inversión de Control, es un patrón en el que el usuario especifica respuestas deseadas a distintos eventos o solicitudes de datos concretos, dejando que alguna entidad o arquitectura externa lleve a cabo las operaciones de control en el orden necesario y para el conjunto de sucesos que tengan que ocurrir. Es decir, los objetos no serían los encargados de realizar el flujo de control, si no que lo delegan en su entorno, siendo éste el que controla lo que ocurre.

Puesto que el principio de inversión de control le otorga el control de la clase al marco de trabajo (a.k.a. *framework*), a la hora de componer dos marcos de trabajo, ambos tendrían el control de la clase simultáneamente, provocando conflictos, ya que realizan cambios de estado sin tener en cuenta la existencia del otro marco de trabajo, haciendo imposible que éstos funcionen juntos.