DP2 2021-2022

Knowledge of WIS' architecture

Repositorio:

https://github.com/mpadillatabuenca/Acme-Toolkit.git

Miembros:

- José Manuel Bejarano Pozo (josbezpoz@alum.us.es)
- Mario Espinosa Rodríguez (maresprod5@alum.us.es)
- Andrea Meca Sánchez (andmecsan@alum.us.es)
- Manuel Padilla Tabuenca (maresprod5@alum.us.es)
- Ezequiel Pérez Sosa (ezepersos@alum.us.es)
- Javier Terroba Orozco (javteroro@alum.us.es)

GRUPO G1-E2-05

Versión 1.0.0

28/02/2022

Tabla de contenidos

Historial de versiones

Fecha	Versión	Descripción de los cambios
28/02/2022	1.0.0	Creación y finalización del documento.

Introducción

En este documento, realizado por José Manuel Bejarano Pozo, hablaré de forma individual sobre los conocimientos previos a la asignatura de Diseño y Pruebas II en cuanto a la arquitectura de los sistemas de información.

La estructura de los contenidos será de la forma:

- **UML:** Definición y tipos de diagramas UML..
- **Diagrama de Componentes:** Definición y funcionamiento de un diagrama de componentes.
- **Diagrama de Clases:** Definición y funcionamiento de un diagrama de clases.
- **Diagrama de Despliegue:** Definición y funcionamiento de un diagrama de despliegue.
- **Diagrama de Secuencia:** Definición y funcionamiento de un diagrama de secuencia.

UML

Una vista representa un aspecto parcial de una arquitectura software. Cada participante en el desarrollo estará interesado en una o varias vistas. El lenguaje unificado de modelado, conocido por sus siglas en inglés como UML (Unified Modeling Language), es un lenguaje gráfico de modelado de sistemas software. Está compuesto por distintos tipos de diagramas que pueden dividirse en dos tipos principales:

✓ Estructurales: muestran la estructura estática de los elementos de un sistema. Entre otros:

- Diagrama de paquetes.
- Diagrama de componentes.
- Diagrama de clases.

- Diagrama de objetos.
- Diagrama de despliegue.

✓ Comportamiento: muestran el comportamiento dinámico de los elementos de un sistema. Entre otros:

- Diagrama de casos de uso.
- Diagrama de actividades.
- Diagrama de secuencia.

En el modelo de vistas “4+1”, el diagrama de componentes y de paquetes representa a la vista de desarrollo, el diagrama de clases a la vista lógica, el diagrama de actividad a la vista de proceso, el diagrama de despliegue a la vista física y el diagrama de casos de uso a la vista de escenarios.

Diagrama de Componentes:

Describen el sistema como una agregación de componentes. Un componente es una unidad software que puede ser reemplazada y actualizada de forma independiente. Los componentes se definen en términos de los servicios que ofrecen (interfaces de salida) y los servicios que requieren de otros componentes (interfaces de entrada).

Se utilizan para representar la arquitectura lógica del sistema. Los elementos de este tipo de diagramas son: componentes, dependencias, interfaces, puertos.
Componentes: buildComponent, Entity, Service, Subsystem.

Interfaces: describen los servicios proporcionados o requeridos por el componente y suelen incluir prototipo de métodos, descripción funcional, limitaciones de uso, etc. Las interfaces de salida (o proporcionadas) detallan los servicios que el componente proporciona, mientras que las interfaces de entrada (o requeridas) describen los servicios que el componente necesita para poder realizar su función.
Caja negra: los detalles de implementación quedan ocultos. Sería lo visto anteriormente.

Caja blanca: se muestran los detalles de implementación del componente, normalmente a través de un diagrama de clases o de otros componentes.

Puertos: los puertos agrupan interfaces relacionadas entre sí. Su uso es más habitual en componentes de caja blanca donde además sirven para relacionar las interfaces con los elementos internos del componente.

Diagrama de Clases:

Se suelen usar con dos fines:

✓ Durante el análisis de requisitos para generar a partir de ellos el esquema de base de datos (IISSI).

✓ Durante el diseño detallado para modelar los principales elementos del código (clases, interfaces,
enumerados, etc.) antes de su implementación (AISS).

Clases: el nombre de las clases debe empezar en mayúscula. Los atributos y métodos se especifican en compartimentos separados.

Atributos: se debe indicar el tipo y la visibilidad (público, privado, paquete, protegido). El nombre debe empezar por minúscula. Se recomienda usar notación camelCase.

Métodos: se debe indicar el tipo de los parámetros y la visibilidad. Los métodos estáticos se indican subrayando su nombre. El nombre debe empezar por minúscula. Se recomienda usar la notación camelCase.

Asociaciones: pueden ser bidireccionales o unidireccionales. Deben incluir la multiplicidad y el rol en cada extremo.

Dependencias: se emplean para indicar que una clase hace uso de objetos de otra clase en cualquier punto de
esta (atributos, parámetros de los métodos, variables locales, etc.).

Herencia: representan la herencia entre clases. Si la clase padre es abstracta su nombre se debe escribir con letra cursiva.

Interfaces: se representan mediante el estereotipo <<Interface>>. Si el editor lo permite, se debe emplear letra cursiva tanto para el nombre como para el cuerpo.

Directrices para el proyecto:

✓ Los atributos y métodos serán opcionales.

✓ Las dependencias se utilizarán para indicar que existe algún tipo de relación entre las clases/vistas.

✓ Las clases se agruparán de acuerdo el paquete lógico al que pertenezcan (modelo, vista, controlador).

✓ Se asignan distintos colores a los distintos paquetes.

✓ Las vistas se modelan también con cajas, como si fueran clases.

✓ El nombre de las vistas empezará en minúscula.

✓ Es necesario indicar la extensión de las vistas.

Diagrama de Despliegue:

Se utiliza para representar la arquitectura física sobre la que un sistema software es desplegado. Por tanto, describe tanto dispositivos hardware como software.

Elementos principales:

✓ Nodo.

- Dispositivo.
- Entorno de ejecución.

✓ Artefacto.

✓ Despliegue.

Nodos: entidades físicas (o software) capaces de ejecutar artefactos. Pueden ser de dos tipos:

✓ Dispositivos (devices): representan elementos hardware, como por ejemplo un servidor, capaces de
ejecutar artefactos.

✓ Entornos de ejecución (execution environment): representan elementos software, como por ejemplo una máquina virtual, capaces de ejecutar artefactos. Siempre deben representarse dentro de un dispositivo.

Un nodo puede contener uno o más nodos en su interior.

Artefacto: cualquier producto obtenido durante el desarrollo:

✓ Ejecutable.

✓ Manual de usuario.

✓ Script de BD.

✓ Librería.

Despliegue: relación entre dos o más artefactos y el/los nodo/nodos donde éstos se ejecutan.

Entorno de ejecución vs. artefacto: un mismo elemento, dependiendo del punto de vista, puede ser considerado como un entorno de ejecución o un artefacto.
Ejemplo:
Eclipse:

✓ Artefacto desde el punto de vista de los desarrolladores de IDE.

✓ Entorno de ejecución desde el punto de vista del desarrollador de uno de sus plugins.

Computación en la nube: permite ofrecer servicios de computación a través de Internet.

Los usuarios pueden contratar y usar los servicios sin necesidad de tener que conocer cómo están implementados. Evita que los clientes (usuarios y empresas) tengan que comprar y mantener su propia infraestructura. El pago por los servicios se suele hacer en función del consumo que se haga de los mismos.

✓ Infrastructure as a Service (IaaS): modelo que ofrece servicios básicos de computación, como servidores, almacenamiento o redes.

✓ Platform as a Service (PaaS): modelo que ofrece entornos avanzados de computación, como bases de datos o entornos de ejecución ya configurados.

✓ Software as a Service (SaaS): modelo que ofrece productos software, como aplicaciones o clientes de correo, listos para ser usados.

Artefacto vs componente: los diagramas de componentes y despliegue están relacionados entre sí mediante componentes y artefactos. Un artefacto manifiesta (o implementa) uno o varios componentes.

Diagrama de Secuencia:

Son un tipo de diagrama de comportamiento. Muestran la interacción de un conjunto de objetos en una aplicación a través del tiempo. Contiene detalles de implementación del escenario:

✓ Objetos y clases usados para la implementación.

✓ Mensajes intercambiados entre los objetos.

✓ Invocaciones a métodos de otros objetos.

Ayudan a mostrar qué se hace y en qué orden. También los usaremos para modelar la interacción entre las distintas aplicaciones integradas en nuestro mashup.
Participantes: son instancias de clases, es decir, objetos. Se especifica la clase a la que instancian. Pueden tener nombre o ser anónimos. La línea discontinua es su línea de vida. En nuestro proyecto los usaremos también para representar ficheros que participen en el flujo de mensajes (ej. formulario.html). Tendrán el color que se le asignó en el diagrama de clases.

Comunicación síncrona: el objeto que envía el mensaje queda bloqueado hasta que recibe la respuesta. Los mensajes se representan con flechas con la cabeza llena. Los

mensajes de respuesta se dibujan con línea discontinua. El texto de las flechas indica el método llamado y lo que se envía como respuesta.

Comunicación asíncrona: los mensajes asíncronos terminan inmediatamente. Se representan con flechas con la cabeza abierta. El texto de las flechas indica el método llamado.

Redirección de flujo de control: para facilitar la comprensión de los diagramas usaremos un tipo distinto de flecha para las redirecciones (no forma parte de la notación estándar). Las representaremos con flechas en azul y con la cabeza abierta. En el texto se indica Forward seguido, entre corchetes, de los parámetros enviados.

Activación: indican cuándo un participante está realizando una acción. Se representan con un rectángulo a lo largo de la línea de vida del participante. Un mismo participante puede activarse y desactivarse varias veces durante su vida.

Alternativas: indican si ejecutar un flujo u otro de acuerdo con una condición.

Flujo opcional: permite ejecutar ciertas acciones sólo si se cumple una condición.
Break: detiene la ejecución si se cumple una determinada condición. Se utilizan para mostrar los casos de error. Representaremos en rojo las flechas dentro de un break (no es parte de la notación).

Bucles: parte del flujo se ejecuta repetidamente mientras se cumpla una condición.

Alto nivel: además de diagramas de secuencia como el anterior, en el proyecto se usa este tipo de diagramas para describir la comunicación entre las aplicaciones integradas.

Conclusión

Las conclusiones obtenidas gracias a la realización de este documento son principalmente, el pequeño resumen de todo lo que he aprendido de la arquitectura de sistemas software gracias a la asignatura de AISS en segundo de carrera.

Con todos estos diagramas he podido llegar a entender de manera correcta cómo funciona un mashup integrado de distintas aplicaciones y espero en esta asignatura de diseño y pruebas 2 aprender mucho más aparte de todo lo aprendido anteriormente.

Bibliografía

1. Apuntes de la asignatura de Arquitectura e Integración de Sistemas Software (ev.us.es)