Resumen Teoría

Sistemas Distribuidos

Melanie Mariam Cruz Morgado

GRADO EN INGENIERÍA INFORMÁTICA

TEMARIO

TEMA 1 3

Paradigamas de computación 3

Sistemas operativos en red 3

Sistemas operativos distribuidos 3

Middleware 3

Modelos arquitectónicos 3

Cliente/Servidor 3

Peer-to-Peer 4

SOA (Arquitectura Orientada a Servicios) 4

MOM (Middleware Orientado a Mensajes) 4

Clúster y Grid 5

Mecanismos de comunicación 5

Mecanismos IPC 6

Protocolos de aplicación 6

Paso de mensajes 6

RPC (llamadas a procedimientos remotos) 7

RMI (Invocación de Métodos Remotos) 7

Intermediario de petición de objetos 7

Decisiones de diseño 7

TEMA 2 8

Tecnologías web. Modelos Básicos y ampliaciones 8

Modelo Cliente/Servidor 8

MIME (Multipurpose Internet Mail Extension) 9

Modelo CGI (Common Gateway Interface) 9

Aplicación Web 9

Middleware 10

Punto de vista de la arquitectura 11

Punto de vista del programador 11

Servicios Web 12

SOAP (Simple Object Access PRotocol) 12

UDDI (Universal Description, Discovery and Integration) 12

WSDL (Web Service Description Languange) 13

Relación entre WSDL, SOAP y UDDI 13

TEMA 3 14

Servicio de nombres 14

DNS (Domain Name System) 14
Servicio de directorio 15
LDAP (Lightewight Directory Access PRotocol) 15
JNDI 16

TEMA 1

PARADIGAMAS DE COMPUTACIÓN

Un **sistema distribuido** (SD) es un conjunto de elementos de computación independientes e interconectados, que comunican y coordinan sus acciones a través de una red de comunicaciones. Ejemplos: Internet, intranets privadas, computación ubicua.

La **computación distribuida** es aquella que se desarrolla en un sistema distribuido: servicios y aplicaciones de red.

Características de un SD:

- **Heterogeneidad**. Rango característico de los paradigmas de computación: estandarización, representación de datos, representación de código, representación de objetos y protocolos.
- **Extensibilidad**.
- **Escalabilidad**.
- **Seguridad**. Entornos que puedan ser atacados, confidencialidad, integridad, disponibilidad, Firewalls, SSL, HTTPS, Radius, Kerberos.
- **Concurrencia y sincronización**: Frente a fallos y de acceso, ubicación, movilidad y escalabilidad.
- **Tolerancia a fallos**.
- **Transparencia**.

SISTEMAS OPERATIVOS EN RED

Tener implementado un SO en red que implica, como **ventajas**, flexibilidad y técnicas maduras, pero tiene la **desventaja** de falta de transparencia y mayor esfuerzo de integración. Ejemplos: Linux, Windows y Novell NetWare.

SISTEMAS OPERATIVOS DISTRIBUIDOS

Tener implementado un SO distribuido implica las **ventajas** de transparencia, escalabilidad y facilidad de integración (características de un SD). Las **desventajas** son que las técnicas son complejas, hay mucha competencia de mercado y se necesitan comunicaciones de alta velocidad. Ejemplos: Mach y Amoeba.

MIDDLEWARE

Tiene un enfoque mixto: el modelo conceptual está en SOD y las infraestructuras en SOR. Es la capa por encima del SO y es homogéneo. Las principales **ventajas** son la flexibilidad, transparencia, integración, madurez y escalabilidad. Las **desventajas** son las plataformas heterogéneas y la necesidad de estandarización.

MODELOS ARQUITECTÓNICOS

CLIENTE/SERVIDOR

Este paradigma asigna roles diferentes a dos procesos que colaboran:

- Proceso **servidor**. Interpreta el papel de proveedor de servicio, esperando de forma pasiva la llegada de peticiones y se encargará de escuchar y aceptar dichas peticiones.
- Proceso **cliente**. Invoca determinadas peticiones al servidor y aguarda respuestas.

Este modelo proporciona una abstracción eficiente para facilitar los servicios en red. Por lo tanto, la sincronización se simplifica: el servidor espera peticiones y el cliente espera respuestas.

PEER-TO-PEER

Los procesos participantes interpretan **ambos roles**, por lo que, tienen idénticas capacidades y responsabilidades. Es decir, cada participante puede solicitar una petición a cualquier otro participante y recibir respuesta de este.

Este paradigma resulta apropiado para aplicaciones como mensajería instantánea, compartición de archivos, vídeo conferencia y trabajo colaborativo.

(*) Es posible que una aplicación se base en ambos modelos: Cliente/Servidor y Peer-to-Peer.

SOA (ARQUITECTURA ORIENTADA A SERVICIOS)

Es un paradigma de arquitectura para desarrollar SD. Permite la abstracción de acceso a actividades de negocio, conocidas como servicios.

Características principales: localización, descubrimiento, publicación, interoperatividad, composición, autonomía, autocontenidos, reusabilidad, desacoplamiento, contrato bien definido y sin estado.

El funcionamiento se basa en que un proceso proveedor publica los datos en un proceso registro, para que un proceso consumidor pueda recurrir a consumirlos. El proceso consumidor realizará una búsqueda del proceso proveedor. El registro enviará la información necesaria al servicio de consumidor para realizar la conexión y ya podrá consumir el recurso del proveedor directamente.

Esquema:

1. **Publicación**. El proveedor se registra en el registro.
2. **Búsqueda**. El consumidor busca al proveedor en el registro.
3. **Descubrimiento**. El registro proporciona al consumidor los datos del proveedor.
4. **Consumo**. El consumidor realiza la petición directamente al proveedor.

Tecnologías basadas en este modelo: JINI, UPNP y Servicio Web (no ha surgido como SOA, pero se acopla perfectamente).

MOM (MIDDLEWARE ORIENTADO A MENSAJES)

Este modelo es una evolución del sistema de paso de mensajes convencional. Permite que la comunicación entre el emisor y el receptor se produzca de una manera completamente desacoplada. Se utiliza en sistemas asíncronos, es decir, donde el emisor no necesita una respuesta inmediata del receptor tras enviar el mensaje y puede continuar funcionando. Elementos: proceso intermediario, proceso emisor y proceso consumidor.

El **proceso intermediario** es el encargado de almacenar los mensajes del emisor, generalmente este es un middleware que incluye ciertos servicios para el tratamiento de estos mensajes. Puede poseer la gestión de prioridades de mensajes, temporizadores para la gestión de mensajes, gestión de formatos de mensajes, gestión de seguridad y gestión de persistencia de los mensajes.

Es un modelo adecuado para aplicaciones y sistemas donde se quiere establecer un único punto de entrada a una comunicación asíncrona de los mismos.

Variantes:

- **Punto a punto** (1:1). Cada mensaje enviado por el emisor únicamente será procesado por un proceso consumidor y el agente intermediario obtiene el mensaje, lo procesa y lo elimina.
- **Publicación/Subscripción** (1:M). Un mensaje publicado por un emisor será procesado por todos los agentes consumidores que se hayan subscrito a dicho proceso intermediario.

Debido a que MOM se apoya en un proceso intermediario para la gestión de los mensajes, una comunicación adecuada sería ORB que dispone de un agente intermediario para la gestión de objetos ofreciéndonos la abstracción del acceso a objetos heterogéneos.

Por otro lado, la arquitectura de servicios web también nos podría dar un enfoque MOM con una mayor interoperabilidad a más alto nivel.

DIFERENCIAS ENTRE SOA Y MOM

- MOM es usado en comunicación asíncrona, mientras que SOA es petición/respuesta, es decir, síncrona.
- MOM consigue desacoplar el escenario completo gracias al proceso intermediario, en SOA únicamente se desacopla la localización de los servicios.
- MOM puede ser usado para la implementación de la arquitectura SOA.

DIFERENCIAS Y SIMILITUDES ENTRE MOM Y EL MODELO CLIENTE/SERVIDOR CONVENCIONAL

- Mientras que el modelo C/S actúa sobre el paradigma de paso de mensajes, la arquitectura MOM es una elaboración más extensa del mismo.
- En C/S solo existen dos procesos (cliente y servidor), mientras que en MOM nos encontramos un tercero, el proceso intermediario.
- Ambos nos permiten la abstracción del acceso a recursos de red.

CLÚSTER Y GRID

Su objetivo es aprovechar los recursos de sistemas independientes conectados a la red para obtener una infraestructura hardware y software con mayor capacidad de procesamiento y almacenamiento. Se basa en la unión de un conjunto de computadores que se unen para dar forma a un supercomputador.

CLÚSTER

Los componentes se encuentran fuertemente acoplados, consiguiendo **homogeneidad**. Se encuentran en redes locales de alta velocidad. El coordinador decide qué procesos se ejecutan en cada participante (gestor de recursos centralizado) y presentan alta disponibilidad, balanceo de carga, escalabilidad y alto rendimiento.

Existe pérdida de independencia en cuanto al procesador, por lo que puede ralentizar de aplicaciones que se estén ejecutando en el mismo nodo del clúster que está consumiendo del procesador.

Algunas aplicaciones son: servidores web y de aplicaciones, sistemas de información y resolución de problemas con necesidad de supercómputo.

GRID

Los elementos son **heterogéneos** conectados a través de redes de área amplia completamente distribuidas, como Internet, por lo que es más flexible. El coordinador coordina los procesos que se ejecutan en los participantes sin decidir cuál se ejecuta en cada participante. No suele perder independencia de procesamiento y no busca ofrecer una visión única del sistema.

MECANISMOS DE COMUNICACIÓN

Los modelos básicos son unidifusión y multidifusión, y las operaciones pueden ser bloqueantes/síncronas o no bloqueantes/asíncronas.

MECANISMOS IPC

El mecanismo básico de comunicación en los SD. El intercambio de datos se realiza en el nivel de red, mediante el flujo binario. Es importante la existencia de heterogeneidad, en la que todos los elementos de la comunicación se adapten a la representación de su receptor o emisor.

Problema: Debe haber una **representación externa** común.

Soluciones:

- PE (Proceso emisor) adapta los datos antes de enviarlos al sistema del PR (Proceso receptor), aunque no es muy fiable ya que siempre se debería conocer el formato de representación del PR.
- PR adapta el formato del PE a su representación interna y junto con la información enviada se debería incluir información del formado de representación del PE.
- Uso de una representación externa que negocien PE y PR, y de esta forma el PE antes de enviar la información, debe transformarla al formato acordado y el PR, cuando la reciba, debe transformarla de la representación externa a su formato interno. Esta solución es la usada por la mayoría de plataformas middleware actualmente.

Problema: Además son necesarios mecanismos que permitan organizar este tipo de datos de manera que puedan ser transmitidos por la red y reconstruidos una vez recibidos.

Solución: Empaquetamiento de datos o **marchalling** que se compone de dos etapas:

- **Serialización**. Las estructuras o tipos complejos de datos son transformados en un formato que permita transmitirlo a través de la red de comunicaciones para su posterior reconstrucción.
- **Codificación**. Estos se codifican a una representación externa antes de ser transmitidos por la red.

XML es uno de los lenguajes externos de representación estándar más usados.

PROTOCOLOS DE APLICACIÓN

- Basados en texto: HTTP, SMTP o POP3.
- Tipo binario: LDAP y SHCP.
- Solicitud-Respuesta: FTP, HTTP y SMTP.
- Técnicas de comunicación: Pool y Push.
- Orientados a la conexión: HTTP y FTP.
- No orientados a la conexión: DNS y DHCP.
- Con estados: FTP y SMTP.
- Sin estados: HTTP.

PASO DE MENSAJES

Intercambio de información entre procesos mediante mensajes. La interfaz mínima requerida debería permitir el envío y el recibimiento de estos mensajes, por lo que la hace dependiente de la conexión.

SOCKETS

Un socket es un punto de comunicación entre dos agentes, por el cual se puede emitir o recibir información. Este mecanismo permite a los procesos comunicarse y sincronizarse entre sí, normalmente a través de un sistema de bajo nivel de paso de mensajes, que ofrece la red subyacente. La comunicación puede ser tanto en local como en remota, siendo la familia INET la más utilizada.

Esta interfaz de comunicaciones es una de las distribuciones de Berkeley al sistema UNIX, implementándose las utilidades de interconectividad de este SO.

La conexión puede ser mediante: flujo de datos (confiable y ordenada) o datagrama (no confiable y sin orden).

Ejemplo de comunicación por sockets: RPC.

RPC (LLAMADAS A PROCEDIMIENTOS REMOTOS)

Protocolo que permite a un programa ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos. Busca poder invocar un procedimiento ubicado en una máquina remota y se basa en el modelo de protocolo petición/repuesta.

Las llamadas a procedimientos remotos que pueden invocarse desde una máquina cliente pertenecen normalmente a equipos servidores. Estas llamadas son muy utilizadas dentro de C/S, siendo el cliente el que inicia el proceso solicitando al servidor que ejecute cierto procedimiento o función y el servidor envía de vuelta el resultado de la operación al cliente.

RMI (INVOCACIÓN DE MÉTODOS REMOTOS)

Mecanismo ofrecido por Java para invocar un método de manera remota. Proporciona un mecanismo simple para la comunicación de servidores en aplicaciones distribuidas basadas exclusivamente en este lenguaje. (En el caso de requerir comunicación entre distintas tecnologías, debemos usar CORBA o SOAP, en lugar de RMI).

Se caracteriza por la facilidad de su uso. A través de RMI se puede exportar un objeto, con lo que pasará a estar accesible a través de la red, y el programa permanece en espera de peticiones en puerto TCP. A partir de ese momento, el cliente puede invocar los métodos remotos.

Componentes:

- **Interfaz remota**. Definición de los métodos remotos.
- **Objeto remoto**. Implementación de los métodos remotos.
- **Cliente**.
- **Stub y skeleton**. Aplicaciones proxies que reciben las peticiones de los clientes y codifican la información a un lenguaje de representación intermedio (XDR), ordenan la secuencia de datos, realizan el proceso de marshalling, establecen la comunicación, etc.
- **Servicio de nombres**.
- **Servicio de registro**. Registro del objeto remoto.

INTERMEDIARIO DE PETICIÓN DE OBJETOS

Uno de los mecanismos de más alto nivel. Principal objetivo: Establecer la comunicación entre objetos escritos en diferentes lenguajes y diferentes plataformas. Este modelo es la base de la arquitectura CORBA definida por el OMG.

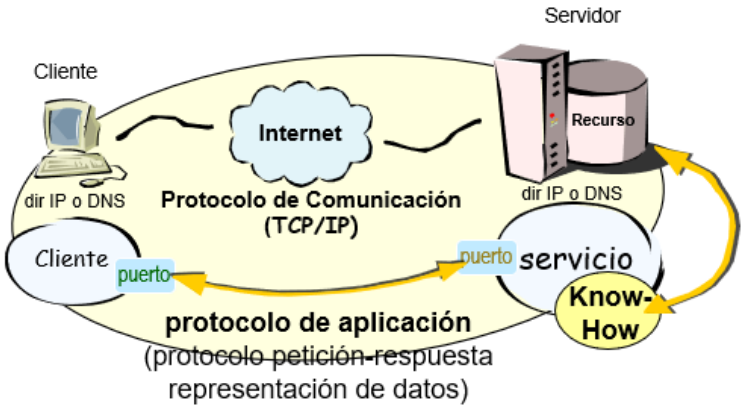
Elementos básicos que conforman esta arquitectura:

- ORB. Agente intermediario para la gestión de objetos.
- IDL y CDR. Definición de interfaces y representación externa.
- GIOP. Protocolos de comunicación.

DECISIONES DE DISEÑO

- Usar mecanismos de mayor nivel de **abstracción** que faciliten el desarrollo y mejoren la productividad, pero que pueden producir una sobrecarga en el diseño final.
- **Escalabilidad** que pueden ofrecer los diferentes paradigmas y permitir crecer al sistema de forma transparente sin que para ellos haya que realizar grandes esfuerzos de modificación o rediseño de la aplicación.
- La posibilidad de poder **portar** las aplicaciones a otras plataformas o la integración de la aplicación con aplicaciones en diferentes lenguajes y plataformas, mediante los paradigmas o herramientas adecuadas.
- **Madurez** y **disponibilidad** de herramientas de desarrollo.
- **Tolerancia a fallos** ofrecida por la herramienta.
- **Mantenibilidad** y **reutilización** de código.

MODELO CLIENTE/SERVIDOR



HTTP

Protocolo de transferencia de hipertexto, usado para la comunicación entre un navegador y un servidor web. Cada petición de cliente especifica un método que deberá ser aplicado en el servidor. Estos métodos son:

- **GET**. Pide el recurso cuyo URL le da como argumento y el servidor siempre devuelve el mismo tipo de dato enviado en la petición URL del cliente.
- **Head**. Petición idéntica a GET, pero no devuelve datos, simplemente toda la información sobre el dato.
- **Post**. Está diseñado para proporcionar un bloque de datos a un proceso de gestión de datos y especifica el URL de un recurso que puede tratar los datos aportados por la petición.
- **Pat**. Indica que los datos aportados en la petición deben ser almacenados con la URL aportada como su identificador.
- **Delete**. El servidor borrará el recurso identificado por el URL. Este método no siempre se permite.
- **Options**. El servidor proporciona al cliente una lista de métodos aplicables a una URL y sus requisitos.
- **Trace**. El servidor envía de vuelta el mensaje de petición y se utiliza para la depuración.

Este protocolo devolverá unos códigos de respuesta:

- **1XX**. Mensajes de información.
- **2XX**. Operación exitosa.
- **3XX**. Redirección hacia otra URL.
- **4XX**. Error por parte del cliente.
- **5XX**. Error por parte del servidor.

Para el mantenimiento de la sesión, podemos utilizar algunos mecanismos como:

- Del lado del servidor → Ficheros, bases de datos y sistema centralizado (sobrecarga).
- Del lado del cliente → Cookies y campos ocultos.

COOKIES

Pequeñas partes de información que se almacenan en el disco duro del visitante a través de su navegador, a petición del servidor de una página. Tiene la ventaja de que se almacena en el usuario y que tienen fecha de caducidad para evitar la sobrecarga.

Una cookie no identifica a una persona, sino a una combinación de PC y navegador. Es importante, para una cookie, conseguir información de hábitos de navegación de usuario.

El funcionamiento se basa en que, cuando un cliente realiza una petición al servidor, este le envía información en la cabecera HTTP, junto a la respuesta, para la creación de cookies en el cliente, mediante *set-cookie*. Para acabar la sesión, el servidor debe establecer una cookie de duración cero.

CAMPOS OCULTOS

Los valores de estos elementos deben ser transferidos mediante atributos *value* y, normalmente, contienen datos para transferir el estado de la sesión.

MIME (MULTIPURPOSE INTERNET MAIL EXTENSION)

Estándar para enviar mensajes de correo electrónico, compuestos por varias partes, conteniendo a la vez texto, imágenes y sonido. Los recursos considerados como datos se proporcionan de forma de estructuras de tipo MIME, tanto en los argumentos como en los resultados, y los datos van precedidos por su tipo de MIME, para que el receptor pueda saber gestionarlos.

MODELO CGI (COMMON GATEWAY INTERFACE)

Protocolo que proporciona una interfaz o pasarela entre un servidor de información y un proceso externo.

Un cliente web puede especificar un programa conocido como script CGI, como objeto web de destino de una petición HTTP, y el servidor web llama al script para activarlo como un proceso, pasándole los datos de entrada transmitidos por el cliente web. El script web ejecuta y transmite su salida al servidor web, que devuelve los datos generados por dicho script.

Ventajas: Capacidad de respuesta dinámica y libre elección del lenguaje de programación.

Desventajas: No hay relación entre el programa CGI y el servidor web, ni control sobre la ejecución, se necesita una nueva instanciación por cada solicitud y existe sobrecarga de recursos.

La principal justificación de su aparición es la necesidad de ofrecer contenido web dinámico, capaz de interactuar con el usuario a través de una aplicación web sobre el protocolo HTTP.

APLICACIÓN WEB

Aplicación basada en la arquitectura C/S, donde el cliente es el navegador y el servidor es el servidor web, utilizando protocolos de Internet (TCP/IP) para su entendimiento.

AMPLIACIONES EN EL CLIENTE

El uso de *scripts* en el cliente es una técnica utilizada para dotar de dinamismo las páginas web.

Ventajas: Máxima fiabilidad, control total sobre el aspecto final y funcionalidad asegurada.

Desventajas: Pérdida de compatibilidad e imposibilidad de obtener *Know-how* debido a la plataforma o versión no válida.

AMPLIACIONES EN EL SERVIDOR

- **Servlets de Java**.

  Programas que se pueden invocar desde el cliente de forma similar a los CGI. El servidor actúa como contenedor de contenedores. Realizan una tarea que se recoge en el servidor web y luego es enviada al navegador.

Características:

- Portabilidad. Solo precisa motor JVM.
- Rendimiento. Única instanciación.
- Sesión. Mantiene información entre diferentes conexiones al servlet.
- Software distribuido. Recuperación de disco/servidor de servlets. Comunicación entre sí.
- Multithread. Múltiples peticiones concurrentemente.

El uso más común es generar webs de forma dinámica a partir de parámetros de una petición enviada por un cliente.

- **Modelo de extensiones (ISAPI/NSAPI).**

  Ventaja: Las librerías de acceso dinámico solo se instancian una vez, ganando rendimiento y aprovechamiento de los recursos.

  Desventaja: La tecnología cerrada a fabricante y servidor web (ISAPI/Microsoft/ISS o NSAPI/Netscape/NES).

- **Modelo de páginas activas.**

  Mecanismo muy extendido que se basa en fragmentos de código insertados en páginas HTML, que el servidor se encarga de interpretar antes de servir dichas páginas. Se realiza con el fin de dinamizar las páginas.

  Esta es la forma de desarrollar más común actualmente, en parte, por la amplia alternativa de lenguajes dependientes para utilizar con la plataforma.

  Ventajas: Habilita el concepto de sesión de usuario, buena gestión de recursos externos, control sobre el protocolo petición/respuesta y operaciones ejecutadas en el servidor.

  Desventajas: Necesidad de conocer lenguaje script.

- **Modelo mixto.**

  AJAX: Tecnología de desarrollo web en conjunto con otras tecnologías existentes. Aplicaciones ricas por parte del cliente, con un buen aspecto.

  JavaScript: Se basa en la ejecución en el cliente.

  XML: Conexión asíncrona con el servidor.

MIDDLEWARE

Nivel lógico del sistema que proporciona una abstracción en términos de servicios, cuya finalidad es proporcionar una visión única del sistema, independiente de la infraestructura que lo forme. Abstrae de la heterogeneidad y complejidad de las redes de comunicaciones, SO y lenguajes de programación.

Aparece ante la necesidad de gestionar grandes aplicaciones sobre entornos distribuidos altamente heterogéneos. Introducción de un nivel de abstracción que aporte independencia con respecto a la plataforma sobre la que se ejecutan las aplicaciones. Permite utilizar tecnologías con un mayor control para la asignación de recursos.

Objetivo: Eliminar la carga del servidor web, colocándose detrás del mismo.

Funcionamiento:

- Las peticiones sobre contenidos estáticos son resueltas por el propio servidor web.
- Las solicitudes de contenidos dinámicos se delegan en el servidor de aplicaciones (Middleware).

PUNTO DE VISTA DE LA ARQUITECTURA

Aplicaciones en términos de componentes. Los elementos fundamentales para este punto de vista son:

MODELO DE REPRESENTACIÓN DE DATOS

Las estructuras de datos y los atributos de los objetos deben ser convertidos en una secuencia de bytes antes de su transmisión y reconstruirse después en el destino. Para que dos computadores intercambien información deben acordar previamente un esquema común, la representación externa de datos.

MODELO DE COMUNICACIÓN

El protocolo petición-respuesta define el nivel adecuado para establecer una comunicación típica según la arquitectura C/S. El propio mensaje es la petición, para evitar sobrecargas. Sobre esta capa se realiza la llamada a procedimientos y la invocación remota de métodos.

PUNTO DE VISTA DEL PROGRAMADOR

MÓDELO DE COMPONENTES

Ofrece reutilización de código. Transparencia con respecto a la plataforma sobre la que se ejecutan y al lenguaje de programación. Capacidad de personalización a través de las propiedades. Comunicación transparente entre ellos y con el contexto, mediante eventos. Tipos de componentes: Cliente y Servidor, encapsulado de servicios o almacén de datos.

J2EE

Plataforma middleware basada en Java, que ofrece 3 tipos de tecnologías para el desarrollo de componentes:

- **Componentes Web**. Responden a una solicitud HTTP.
  - Servlets. Programas Java que componen páginas web.
  - JSP. Páginas web que contienen código en Java.

- **Componentes Enterprise JavaBeans (EJB).** Modelo de componentes distribuidos. Unidades de software reusables que contienen lógica de empresa. Tipos de Beans:
  - De sesión. Ejecuta solicitudes del cliente y es destruida cuando se completan. Se mantiene el estado entre solicitud y solicitud (opcional).
  - De entidad. Objeto persistente que modela los datos de un almacén.
  - Dirigidos por mensaje. No son invocados por el cliente y procesan mensajes asíncronos.

Además, establece 4 tipos de contenedores:

- **Contenedor Web**. Para servlets y JSP.
- **Contenedor EJB**. Componentes Enterprise JavaBeans.
- **Contenedor Applet**. Para aplicaciones Java ejecutadas en navegadores.
- **Contenedor de aplicaciones cliente**. Para aplicaciones Java estándar.

SERVICIOS WEB

Procesos o funciones de negocio significativas con una interfaz bien definida y accesible a través de Internet, basada en el intercambio de mensajes en XML, pudiendo ser estos procesos o funciones combinados entre sí.

El paradigma utilizado para el desarrollo de SD es B2B. Proporciona sistemas débilmente acoplados, permite la reutilización y composición, y la interoperabilidad está basada en contratos bien definidos.

La comunicación entre los negocios a nivel de aplicaciones se produce mediante la utilización de estándares abiertos como XML, SOAP, UDDI y WSDL.

SOAP (SIMPLE OBJECT ACCESS PROTOCOL)

Protocolo ligero basado en XML, para el intercambio de la información en un entorno descentralizado y distribuido. Digamos que es el protocolo sobre el que hablan los servicios. Este deriva de XML-RPC y nos facilita la comunicación entre objetos de cualquier tipo, sobre cualquier leguaje y plataforma. Además, permite la comunicación máquina a máquina débilmente acoplada y el intercambio de mensajes a través de firewalls.

Consta de las siguientes partes:

- **Sobre o envoltura**. Define un marco de referencia general para expresar qué hay en el mensaje, quién debe atenderlo y si es opcional u obligatorio. Identifica un mensaje XML como SOAP.
- **Reglas de codificación**. Definen un mecanismo de serialización, que se puede utilizar para intercambiar instancias de tipos de datos definidos para la aplicación.
- **Representación RPS/Document**. Define una convención que puede ser utilizada para representar las llamadas y respuestas a procedimientos remotos.

MENSAJES SOAP (XML (EXTENSIBLE MARKUP LANGUAGE))

Solución para el intercambio de datos de una forma transparente.

Estos mensajes tienen dos partes:

- **Recubrimiento**. Definido por el elemento *<SOAP-ENV: envelope>* con un conjunto de atributos requeridos, que especifican el esquema de codificación y el estilo del recubrimiento.
- **Cuerpo**. Definido por la etiqueta *<SOAP-ENV: body>***.** Esta parte solo contiene un elemento, la llamada del método con sus parámetros.

UDDI (UNIVERSAL DESCRIPTION, DISCOVERY AND INTEGRATION)

Mecanismo que nos permite localizar y saber con quién comunicarse para acceder a un servicio web concreto y dónde hacerlo. Objetivo: Ser accedido por los mensajes SOAP y dar paso a documentos WSDL.

Categorías de API:

- **De publicación**. Para que los proveedores de servicios se registren (ellos y sus servicios) en el registro UDDI.
- **De consulta**. Permite, a los subscriptores, buscar los servicios disponibles y obtenerlos una vez localizados.

Secciones conceptuales de un registro UDDI:

- **Blanca**. Similar a la información que aparece en el directorio telefónico. Incluye nombre, teléfono y dirección.
- **Amarilla**. Similar a su equivalente telefónico. Incluye categorías de catalogación industrial tradicionales, ubicación geográfica, etc.
- **Verde**. Información técnica acerca de los servicios ofrecidos por los negocios.

Estructura central de un registro UDDI:

- **businessEntity**. Información sobre un negocio o entidad. Utilizada por el negocio para publicar información descriptiva sobre sí mismo y los servicios que ofrece.
- **businessService**. Servicios o procesos de negocios que provee la estructura businessEntity.
- **bindingTemplate.** Datos importantes que describen las características técnicas de la implementación del servicio ofrecido.
- **tModel.** Especificación y categorización técnica.

WSDL (WEB SERVICE DESCRIPTION LANGUANGE)

Formato XML utilizado para describir los mensajes SOAP que definen un servicio web en particular. Es un IDL (Interface Definition Language) para la comunicación con el servicio y definir de manera abstracta una interfaz pública del mismo. Describe principalmente el protocolo a utilizar y la implementación concreta del servicio.

Anatomía:

- **<definitions>** . Definición de uno o más servicios.
- **<messages> y <portType>.** Operaciones que provee el servicio.
- **<binding>.** Cómo se invocan las operaciones.
- **<service>.** Dónde se ubica el servicio.
- **<documentation>.** Puede contener información del servicio para el usuario.

Estructura:

- **Types**. Definiciones de los tipos de datos para describir los mensajes intercambiados.
- **Message**. Definición abstracta de los datos que se transmiten. Un mensaje es dividido en partes lógicas, cada una asociada a una definición de sistema de tipos.
- **PortType**. Operaciones abstractas que hacen referencia a un mensaje de entrada y uno de salida.
- **Binding**. Especifica el protocolo concreto y las especificaciones del formato de datos de los mensajes definidos por un portType concreto.
- **Service**. Unir un conjunto de puertos relacionados.
    - **Port**. Dirección para un Binding, para definir un único nodo de comunicación.

RELACIÓN ENTRE WSDL, SOAP Y UDDI

La relación entre estas tecnologías radica en que son los estándares o protocolos que permiten la comunicación entre los negocios a nivel de aplicaciones, contando de las partes de: publicación, búsqueda, descubrimiento y consumo.

TEMA 3

SERVICIO DE NOMBRES

Servicio que provee a los clientes información sobre elementos de un SD de manera legible para el ser humano, con el fin de identificar los mismos. Es usado para referenciar recursos y usuarios, así como comunicar y compartir recursos. Este servicio almacena colecciones de pares <nombre, atributo> y busca atributos a partir de nombres. (Símil con las páginas blancas).

Características:

- Utilizan el paradigma C/S.
- Servicio independiente fácilmente escalable.
- Independencia de su ubicación.
- Alta disponibilidad.
- La información se almacena jerárquicamente.
- Débil consistencia de replicación.
- Flexibilidad.
- Base de datos optimizada, orientada a la lectura de información, datos de una entrada en un único registro, no necesita transacciones y tampoco bloqueos.

DNS (DOMAIN NAME SYSTEM)

Es un servicio de nombre, ya que nos permite buscar un atributo a través de un nombre, y específico debido a que nos permite crear un espacio y extender la funcionalidad, guardando la información en la estructura creada.

Definición:

- Establece una jerarquía de nombres para nodos en redes TCP/IP.
- Asocia cada nombre con una dirección IP.
- Todo un sistema basado en BD distribuida que permite resolver (directa e inversamente) nombre DNS a dirección IP y viceversa.

Elementos:

- **Espacio de nombres**. Jerarquía estructurada de dominios para organizar los nombres.
- **Registros de recursos**. Asignan nombres a un tipo específico de información de recurso (utilizada para resolver el nombre en el espacio de nombres).
- **Servidores DNS**. Almacenan y responden a las consultas de nombres.
- **Clientes DNS**. Consultan a los servidores para buscar y resolver nombres de un tipo de registro de recursos.

Funcionamiento básico:

1. Petición del cliente.
2. El cliente DNS solicita la resolución de un nombre.
3. El servidor DNS revuelve la IP asociada al nombre.
4. El cliente puede realizar su petición.

Espacio de nombres:

- **Dominio**: Agrupación lógica del espacio de nombres. Un subárbol del espacio de nombres de dominio → ua.es
- **Subdominio**: Otros dominios dentro de un dominio → desarrollo.ua.es
- **Zona**: Unidad más pequeña y manejable, creada por delegación. Relacionada con la gestión y resolución de recursos. Normalmente es un archivo físico que gestiona un conjunto de recursos y puede que de varios dominios.

SERVICIO DE DIRECTORIO

Servicio que provee a los clientes información sobre objetos que satisfacen una determinada descripción. Es similar al servicio de nombres, pero este nos permite buscar nombres a través de atributos. (Símil con las páginas amarillas).

Características:

- Información acerca de objetos relacionados (recursos de red, personas…).
- Refuerza la seguridad para proteger a los objetos de intrusos.

LDAP (LIGHTEWIGHT DIRECTORY ACCESS PROTOCOL)

Es un servicio de directorio, ya que nos permite realizar búsquedas por atributo y/o distinta información. Además, es de propósito general, ya que solo nos permite crear un espacio de nombres.

Características:

- Puede **utilizar bases de datos** para recuperar información de forma rápida o hacer una consulta de los datos.
- Puede **dividir el árbol de directorios en subárboles gestionados por diferentes servidores LDAP**. Distribuye, en la red, información lista para ser usado por todas las aplicaciones. Todo esto sin afectar ningún acceso externo a estos datos.
- **Independencia de la plataforma**. Como LDAP es un protocolo estándar, que proporciona un método de acceso a datos remoto y local, es posible intercambiar completamente la implementación del LDAP sin afectar la forma en que se podrá acceder a los datos. El cliente y el servidor se pueden ejecutar en SO diferentes.
- **Seguridad integrada en el repositorio**. La información de acceso se almacena en el mismo repositorio.
- **Implementación fácil del cliente**. La disponibilidad de interfaces de programación API del LDAP, para casi cualquier lenguaje de programación, facilita la **compatibilidad** del LDAP con prácticamente todas las aplicaciones.
- **Gran difusión**.
- **Bajo coste**. Sin pagar licencias por su uso ni para disponer de clientes o servidores.

Funcionamiento:

En el modelo C/S del LDAP, ante una consulta concreta de un cliente, el servidor contesta con la información solicitada. La respuesta puede ser, de forma alternativa (o además de la información que se había solicitado), un puntero que indica donde conseguir esta información o datos adicionales.

Se puede decidir separar el directorio entre varios servidores, por motivos organizativos o para facilitar su gestión. El servidor está configurado para devolver referencias a otros servidores LDAP, en caso de que se le pida información de la que no dispone, pero que sabe dónde conseguirla.

Se puede aumentar la disponibilidad y la fiabilidad del directorio utilizando más de un servidor LDAP para mantener la información.

Usos empresariales:

- Directorios de información.
- Sistemas de Autentificación/Autorización.
- Sistemas de información de cuentas de correo electrónico.
- Grandes sistemas de autenticación basados en RADIUS.
- Servidores de certificados públicos y llaves de seguridad.
- Perfiles de usuarios centralizados.

Modelos:

- **De información**. Define qué tipo de información o dato (esquemas, entradas, atributos) se puede almacenar en el directorio y las unidades básicas en que el LDAP estructura la información. Describe la estructura de la información almacenada en el directorio LDAP. Utiliza ficheros ASCII para entradas LDAP.
- **De asignación de nombres o nomenclatura**. Define cómo se organiza, se identifica y se referencia la información en el directorio LDAP. Las entradas de directorio se disponen en una estructura de árbol jerárquica. El nombre de una entrada debes ser único en cada servidor LDAP.
- **Funcional**. Define cómo se recupera y modifica la información del directorio, describiendo las operaciones que se pueden realizar en el acceso, el mantenimiento y la gestión del directorio. Describe qué operaciones pueden ser realizadas con la información almacenada en el directorio LDAP.
- **De seguridad**. Muestra cómo se controla el acceso a la información contenida en el directorio. Antes de que un cliente pueda acceder a los datos de un servidor LDAP, se llevan a cabo dos procesos:
    - **Autenticación**. Para asegurar que las identidades de los usuarios y máquinas están validadas.
    - **Autorización**. Para controlar el acceso a los recursos del directorio a las personas o entidades que intentan acceder a ellos).

(*) La justificación de porqué la gestión es distribuida es que de esa forma puede dividirse en subárboles por motivos de rendimiento, localización geográfica y por cuestiones administrativas.

(*) A diferencia del sistema tradicional (X.500):

- LDAP utiliza TCP/IP en lugar de protocolos OSI.
- El modelo funcional de LDAP es más simple y ha eliminado opciones raramente utilizadas en X.500. Por lo que, LDAP es más fácil de comprender e implementar.
- LDAP representa la información mediante cadenas de caracteres en lugar de complicadas estructuras ASN.1.

(*) La técnica utilizada para relacionar las diferentes partes del espacio de nombres es el uso del objeto *ObjectClass:referal*, atributo obligatorio para almacenar la URL de acceso a los subárboles.

JNDI

Esta arquitectura consiste en una API y un SPI (Service Provider Interface). Las aplicaciones Java usan el API JNDI para acceder a una gran variedad de servicios de nombre y directorios. El SPI permite conectar, de forma transparente, una gran variedad de servicios de nombres y directorios, por lo tanto, permite a las aplicaciones Java usar el API JNDI para acceder a sus servicios.

Características:

- JNDI ≈ JDBC.
- Unificar el acceso a SN y SD. Acceso transparente.
- Arquitectura de plugin. Conexión dinámica de diferentes implementaciones.
- Federación. Comunicación entre Proveedores de servicios (LDAP → DNS).

Funciones:

- Interfaz Context (javax.naming).
    - Inicializar el contexto.
    - Buscar (lookup).
    - Unir y desunir (bind y unbind).
    - Renombrar objetos (rename).
    - Crear y eliminar subcontextos (createSubcontext y destroySubcontext).
    - Enumerar enlaces (listBindings).

- Interfaz DirContext (javax.naming.directory).
    - Extiende javax.naming.
    - Acceso a directorios además de nombres.
    - Trabaja con atributos.
    - Obtener y modificar atributos (getAttributes y modifyAttributes).
    - Búsqueda por filtros (search).

- JNDI → rmiregistry.