Capítulo II: Arquitectura en Sistemas distribuidos

Estilos arquitectónicos, patrones y sistemas

Prof. Dr.-Ing. Raúl Monge Anwandter ♦ 2º semestre 2025

Objetivos del capítulo:

Objetivo general:

- Disponer de una marco conceptual, para comprender los diferentes estilos arquitectónicos en sistemas distribuidos y la integración de elementos teórico-prácticos.

Objetivos de aprendizaje:

1. Reconocer en sistemas distribuidos principales estilos arquitectónicos y relacionar algunos patrones de diseño.
2. Comprender características de una arquitectura de un sistema distribuido, sus componentes, conectores y patrones de interacción.
3. Entender principios de descubrimiento y ligado (*binding*) de componentes dinámico en una arquitectura distribuida.

Organización del capítulo:

1. Introducción a arquitectura sistemas distribuidos
2. Estilos arquitectónicos en sistemas distribuidos
3. Nombramiento y descubrimiento de componentes

2.1 Introducción a Arquitectura en Sistemas Distribuidos

Arquitectura en Sistemas Distribuidos

Arquitectura de Software

DEFINICIÓN: Una arquitectura se refiere a la organización y estructura de un sistema, abordando aspectos de diseño de alto nivel, tales como:

- **Componentes.** Corresponden a los bloques de construcción.
- **Relaciones**. Conexiones entre componentes (conectores e interfaces) y con el entorno del sistema.
- **Principios**. Guía de diseño con reglas/restricciones para componentes y conectores, para facilitar la evolución del sistema.

PROPÓSITO:

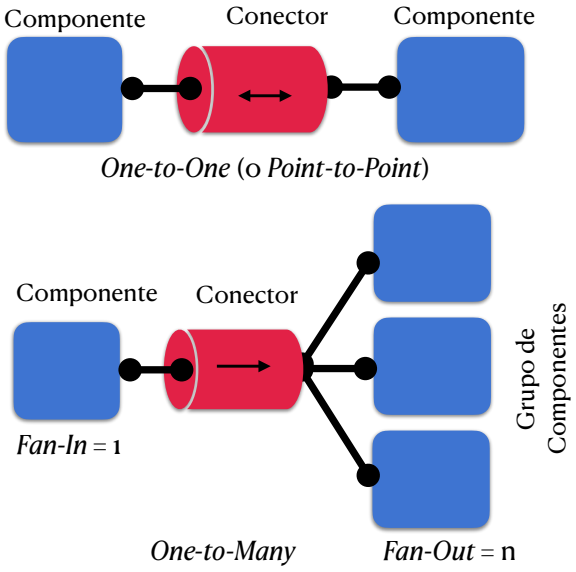
- Definir un plan para construir sistemas robustos, eficientes y mantenibles, que cumplan los requisitos funcionales y no funcionales (NFR).
- Actuar de puente entre el diseño técnico y los objetivos operativos de un sistema, garantizando que su comportamiento se ajuste a ciertos requisitos o restricciones.

Componentes y conectores

DEFINICIONES:

- **Componente**: Unidad modular con interfaces bien definidas, reemplazable y reusable.
- **Conector**: Enlace de comunicación que media en la coordinación o cooperación entre componentes.

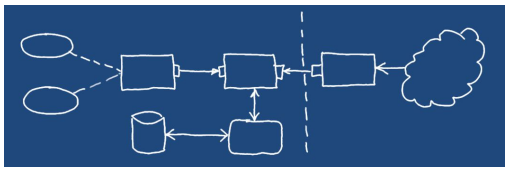
OBS: Se establecen multiples patrones de interacción entre componentes.



Ejemplos de componentes y conectores

Componentes

- Máquinas o procesadores
- Procesos
- Objetos (e.g. Corba, Java RMI)
- Componentes (e.g. DCOM, Contenedor)
- Recursos (e.g. en HTTP)
- Servicios (e.g. en SOA y REST)

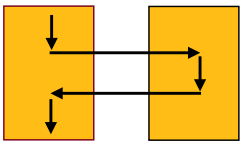


Conectores

- Invocaciones remotas (RPC, RMI, WS)
- Memoria compartida (DSM, Archivos)
- Pipe & streams / Filters (e.g. UNIX)
- Sockets (e.g. TCP o UDP)
- Base de datos (e.g. ODBC)
- Broker de peticiones (e.g. ORB en Corba)
- Broker de mensajes (e.g. ESB)
- Bus de datos (e.g. *event streaming*)

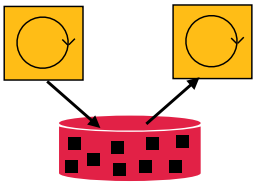
Paradigmas de comunicación

Comunicación entre componentes en programación distribuida



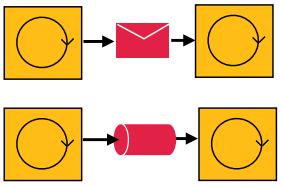
a) Transferencia de control (comando o petición)

- Comunicación directa por paso de parámetros y resultados.
- *Multithreading* requiere control de concurrencia en componentes compartidos.
- Casos de uso: Invocar/llamar un subprogramas; modelo de comandos (e.g. llamada al sistema); extendido en la red para invocaciones y comandos remotos, tb. movilidad de código.



b) Memoria compartida (Repositorio de datos)

- Comunicación indirecta usando un espacio de almacenamiento de datos compartido.
- Requiere control de concurrencia para garantizar consistencia.
- Casos de uso: memoria compartida (e.g. UMA, DSM, espacio de tuplas); repositorio de datos persistentes (e.g., archivos, BD, objetos de datos).

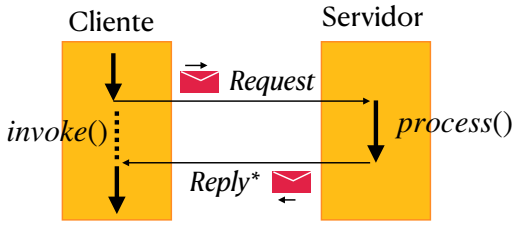


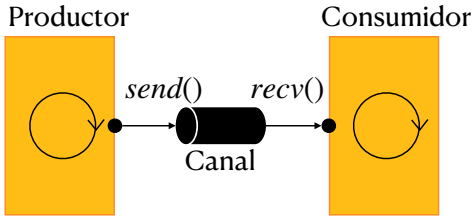
c) Transferencia de datos (mensaje)

- Mensajes son explícitos, con comunicación directa o indirecta (si existe intermediario).
- Dos estilos: Paso de mensajes y *data streaming*. También multipunto.
- Casos de uso: Tradicional para comunicación en redes (e.g. *sockets*): colas de mensajes y notificación de eventos.

a) Estilos de interacción directa

Interacciones sincrónicas entre componentes conocidos





a) Invocación remota

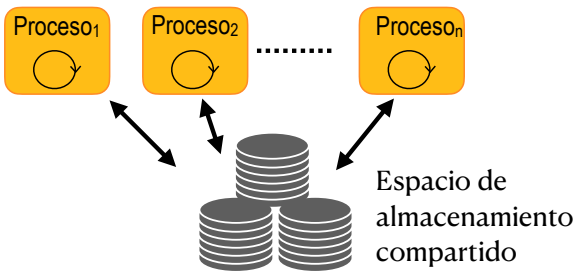
- Comunicación directa normalmente bidirreccional (puede ser múltiple).
- No existe memoria compartida.
- Servidor controla sus recursos.
- Casos de uso: Simula una transferencia de control, para solicitar un servicio remoto.

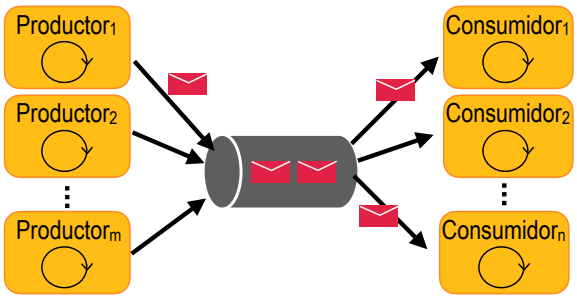
b) *Data streaming* (flujo de datos)

- Comunicación directa por flujo continuo de datos (se vuelve indirecto si canal asume rol de componente intermediario).
- Usualmente se controla el flujo.
- Casos de uso: transferencia de archivos, multimedios y flujo de eventos o comandos.

b) Estilos de interacción indirecta

Interacciones asincrónicas y desacopladas





c) Repositorio de datos compartidos

- Comunicación indirecta a través de un espacio de datos compartido (c/u con dirección conocida).
- Requiere control de concurrencia y recuperación ante errores, para garantizar consistencia de datos.
- Ejemplos: DSM, Archivos, SGBD, Espacio de tuplas.

d) Colas de mensajes (tb. *Streaming*)

- Comunicación indirecta a través de conector intermediario (e.g. buzón, cola, *broker*, bus de mensajes).
- Consumo asíncrono (*put*) o por sondeo (*pull* o *polling*).
- Ejemplos: Message Queue, Pub-Sub, *streaming* de eventos.

Estilo arquitectónico

DEFINICIÓN: Estrategia o *framework* abstracto y de alto nivel, que define la estructura general y los principios organizativos de un sistema.

- Determina para un sistema: ¿cómo interactúan los componentes?, ¿cómo dividir responsabilidades?, ¿cómo escalar?, ¿cómo tolerar fallos?, etc.

Características:

- Define reglas fundamentales (e.g. componentes se comunican a través de eventos).
- Normalmente es tecnológicamente agnóstico (diversas opciones de implementación).
- Orienta sobre decisiones de diseño globales (e.g. centralizado vs. descentralizado).

Ejemplos de estilos arquitectónicos:

- Cliente-Servidor, Microservicios, Peer-to-Peer, Event-Driven, RESTful, Serverless.

Estilo de Arquitectura Cliente-Servidor

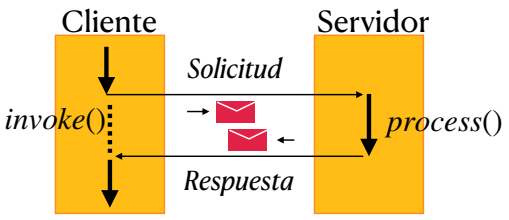
DEFINICIÓN: Divide el sistema en componentes que solicitan (clientes) y componentes que proveen (servidores) un servicio o dan acceso al uso de algún recurso compartido.

- El cliente inicia la acción y el servidor responde procesando una solicitud o petición (*request*).
- Se basa en un modelo o patrón de interacción de tipo *request-response* (solicitud-respuesta)
- Puede ser sin estado (e.g. REST, función) o con estado (e.g. servidor de datos).

Casos de uso:

- Llamar a una función que retorna un resultado
- Solicitar alguna información
- Notificar sobre algún evento, pero esperar asentimiento.

Servicios típicos de apoyo: descubrimiento, seguridad.



Evaluación:

+ Control centralizado (acceso y uso); escalabilidad (particionando o replicando).
– Punto único de falla (resiliencia); posible cuello de botella (desempeño)

Lectura complementaria: [van Steen & et al., 2023] sección 2.3.1: *Simple cliente-server architectures*

Descubrimiento y ligado (*binding*)

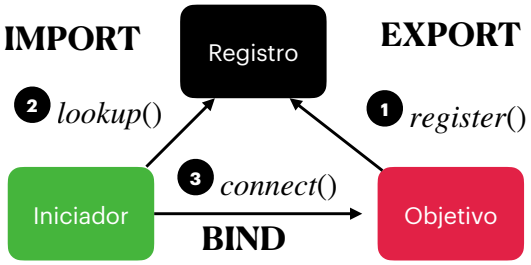
Nombramiento de componentes

Resolución de nombre:

- **Estático**: Está resuelto *a priori* a la interacción, i.e. está predefinida (e.g. puerto 80 para HTTP).
- **Dinámico**: Se resuelve en tiempo de ejecución antes de interactuar, usando nombre preestablecido (i.e. permite soportar configuraciones dinámicas del sistema).

Según estilo de interacción:

- **Interacción directa**: Iniciador debe conocer *a priori* nombre de entidad objetivo.
- **Interacción indirecta**: Entidades activas sólo debe conocer nombre del intermediario (e.g. un conector o *broker*).



Ejemplo: resolución dinámica e interacción directa

OBSERVACIONES:

- Corresponde a "Patrón de Registro" para descubrir y localizar dinámicamente un componente.
- Servicio de registro debiera tener una dirección estática (u otro servicio que termine recursión).

Patrones arquitectónicos (y/o diseño)

Solución reutilizable a problemas comunes en una arquitectura

- **Proxy**[1]. Intermediario entre clientes y un servicio; con *caching*, balance de carga o seguridad (e.g. *stub*, *proxy* inverso).
- **Broker**. Intermediario entre clientes y componentes distribuidos, manejando comunicación, enrutamiento de mensajes y coordinación (e.g. ORB en Corba, ESB en SOA, *Message Broker*).
- **Gateway**[3]. Intermediario que traduce protocolos, API o formatos de datos entre diferentes sistemas. Encapsula acceso a sistemas externos o recursos, proveyendo un único punto de acceso (e.g. API Gateway, SOAP a REST, MQTT a HTTP).
- **Adaptador**[1]. Convierte una interfaz de un componente en otra interfaz esperada por un cliente (JDBC y diferentes BD). Similar a patrón *Wrapper*. Permite reducir el número de interfaces conocidas por el cliente.
- **Fachada**[1]. Interfaz unificada para un conjunto de interfaces en un subsistema, facilitando su uso.
- **Registro**[2]. Componente centralizado que realiza seguimiento de objetos o servicios para gestión de una configuración eficiente.
- **Model-View-Controller** (MVC)[2]. Divide interacción de interfaz usuaria en tres roles diferentes.

FUENTES:
1. E. Gamma, et al. (1994). "*Design Patterns: Elements of Reusable Object-Oriented Software*", Addison-Wesley.
2. M. Fowler (2002). "*Patterns of Enterprise Application Architecture*", Addison-Wesley.
3. G. Hohpe & B. Woolf (2003). "*Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*", Addison-Wesley.

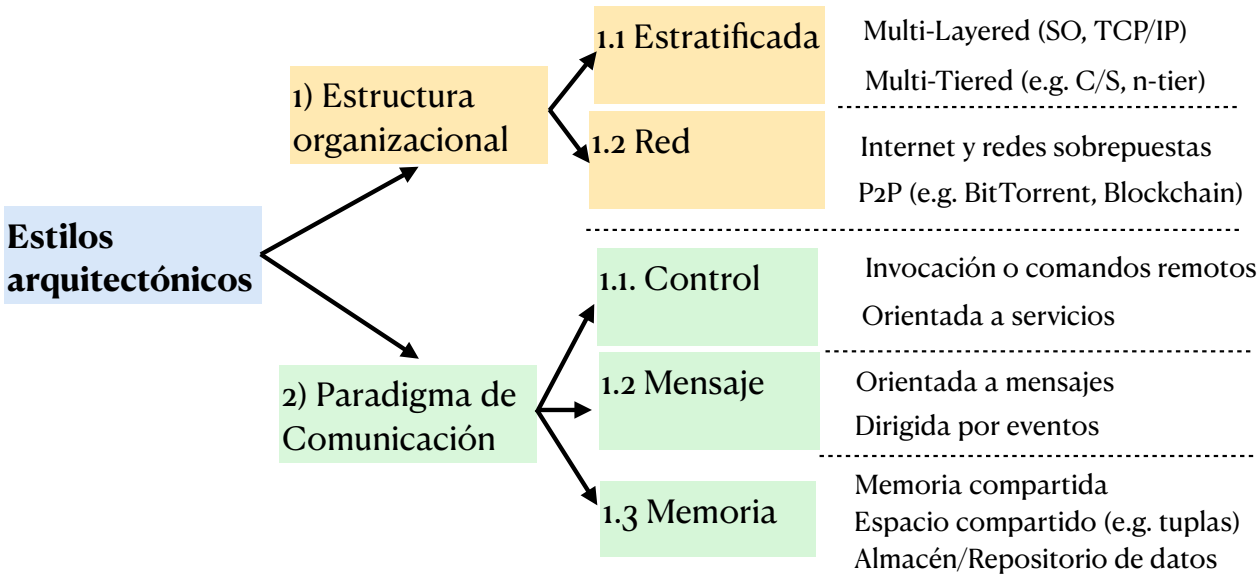
Gestión de cambios de configuración

Un problema fundamental para la gestión de componentes

- **Necesidad de cambios de configuración**
  - Replicación y múltiples proveedores de un servicio
  - Balance de carga
  - Cambios de configuración por elasticidad (*scale-up* o *scale down*)
  - Tolerancia a fallos y recuperación de errores para componentes
  - Mantenibilidad
- **Servicios de registro, nombre y directorio**
  - Descubrimiento, resolución de nombres y direccionamiento
  - Requiere alta disponibilidad y tolerancia a fallos (por ser un servicio crítico)

2.2 Estilos arquitectónicos en sistemas distribuidos

Una taxonomía para estilos arquitectónicos



Estilo arquitectónico Nº1: Arquitectura *Multi-layered*

Lectura complementaria: van Steen (2023). Sección 2.2.1 "*layered architectures*", pp. 57-59.

Arquitectura Multi-*layered* (estratificado)

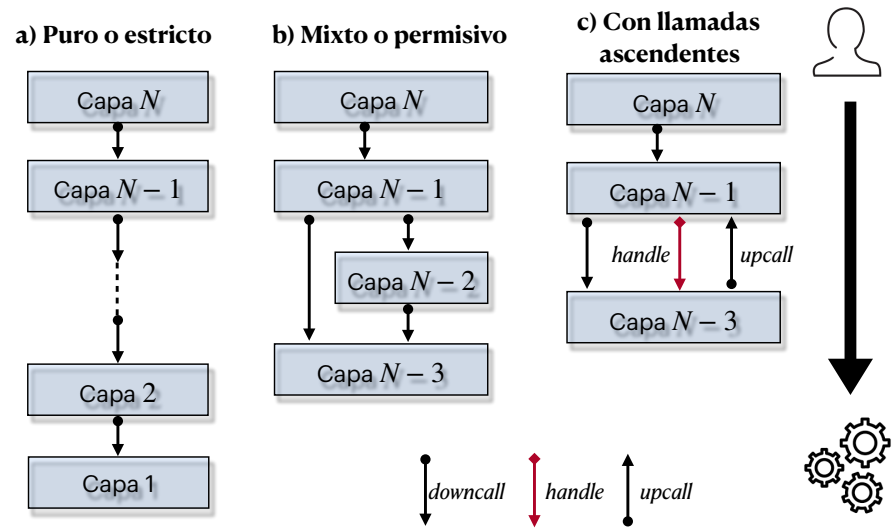
Capas o niveles de abstracción de software (desde el *hardware*)

Características:

- Organiza sistemas complejos separando funciones por valor agregado, reforzando modularización y reusabilidad.
- Arquitectura se organiza por capas o niveles de abstracción (más bajo es el *hardware*).

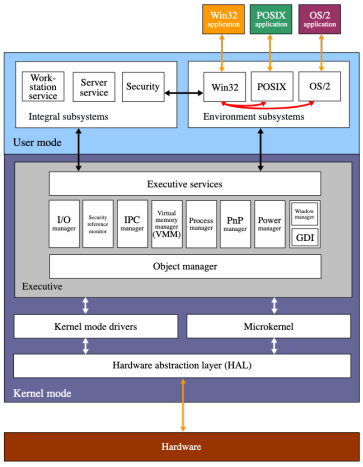
Interacción entre capas:

- Cada capa usa servicios de niveles inferiores (*downcall*).
- Registro de función superior (*handle*) permite recibir llamadas desde capas inferiores (*upcall*).

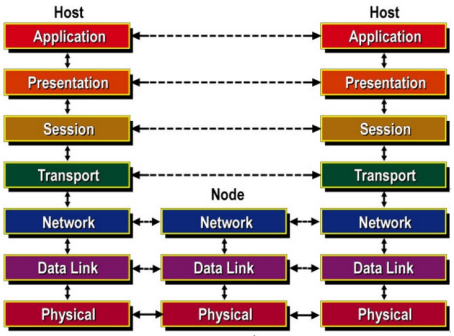


Ejemplo de arquitectura de capas (*layered*)

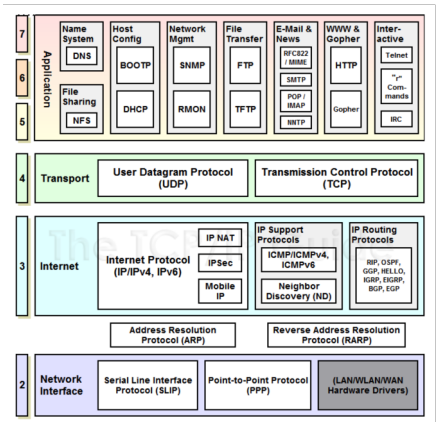
Sistemas operativos y redes de computadores



a) Sistema operativo Windows NT



b) Modelo ISO/OSI



c) Modelo TCP/IP

Ejemplo: Arquitectura de *Middleware*

Servicios y protocolos de un Middleware (Subcapas)

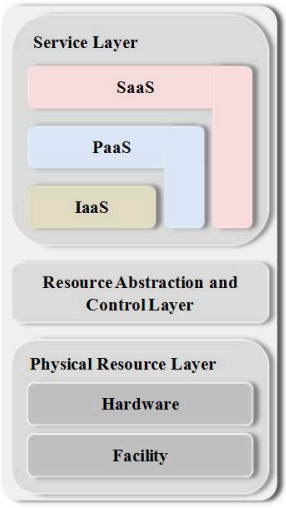
- **Componentes**:
  - **Servicios básicos**: Coordinación, tiempo y comunicación.
  - **Servicios superiores**. Almacenamiento de datos, nombramiento, transacciones, balanceadores de carga, *logging* & monitoreo, seguridad.
- **Conectores y comunicación**:
  - **Transporte de datos** (e.g. *sockets*, transporte basado en HTTP)
  - **Sincrónicos** (e.g. invocaciones remotas, conectores a bases de datos)
  - **Asincrónicos e indirectos** (e.g. Colas/*brokers* de mensajes/eventos)

↓ Mayor valor agregado

↓ Mayor valor agregado

Arquitectura de Servicios Cloud

Principales capas de abstracción (*layers*)



- **Infraestructura como Servicio (IaaS)** . Proporciona a través de Internet recursos virtualizados de *datacenters*.
  - Usuarios administran sistemas operativos, aplicaciones y almacenamiento.
  - Ejemplos: Amazon EC2, Google Compute Engine, MVs de MS Azure.
- **Plataforma como Servicio (PaaS)** . Ofrece una plataforma para desarrolladores para construir, desplegar y gestionar aplicaciones, en entornos preconfigurados.
  - Usuarios implementan aplicaciones sin administrar la infraestructura.
  - Ejemplos: Google App Engine, AWS Lambda.
- **Software como Servicio (SaaS)** . Ofrece aplicaciones completamente funcionales a través de Internet.
  - Usuarios sólo interactúan con el software, sin preocuparse por su mantención.
  - Ejemplos: Gmail, Microsoft 365, Dropbox, Zoom.

Estilo arquitectónico Nº2: Arquitectura *Multi-tiered*

Lectura complementaria: van Steen (2023). Sección 2.3.2 "*Multi-tiered Architectures*", pp. 80-88.

Arquitectura *Multi-tiered* (estratificado)

DEFINICIÓN: Estilo arquitectónico que organiza un sistema en capas lógicas, cada una de ellas con una responsabilidad específica (e.g. presentación, lógica de negocio y datos), con el propósito de definir un patrón de despliegue (físico) en un ambiente distribuido de múltiples máquinas.

Casos de uso:

- Arquitectura cliente-servidor
- Aplicaciones Web y móviles
- Aplicaciones empresariales

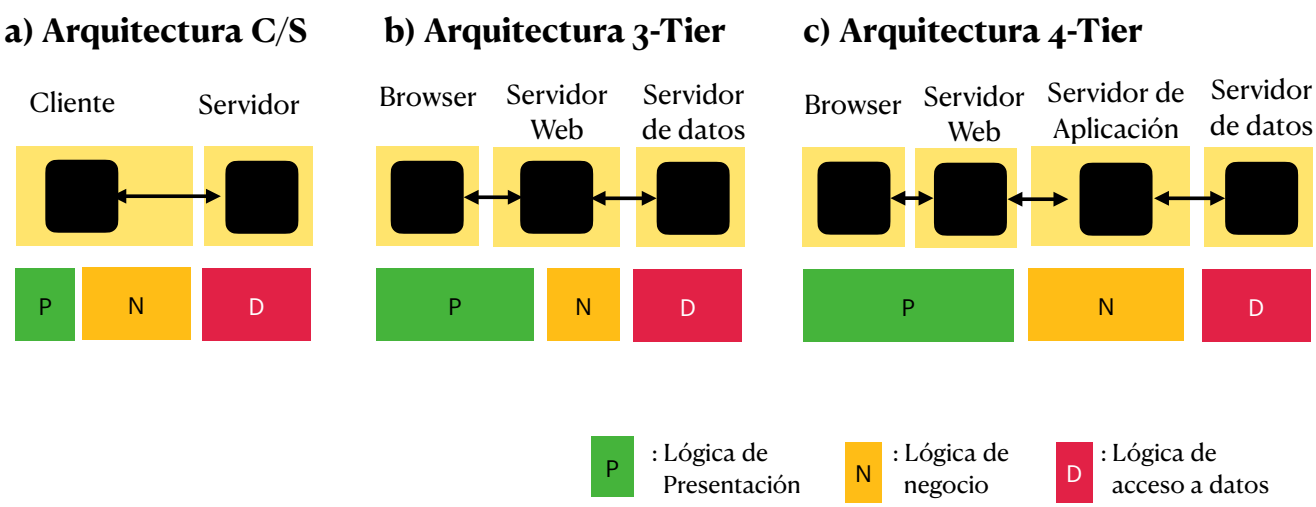
Evaluación:

+ Separación de responsabilidades (*concerns*), modularidad, reusabilidad y escalamiento.
– Puede introducir latencia y sobrecarga; tb. acoplamiento fuerte entre capas (rigidez).

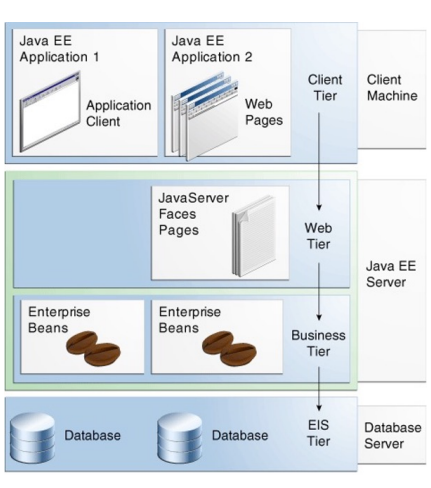
OBSERVACIÓN: Diferentes autores usan "*multi-tiered*" y "*multi-layered*" para referirse al mismo concepto.

- Un punto de vista es ser una *distribución horizontal*, a un mismo nivel de abstracción (*layer*).

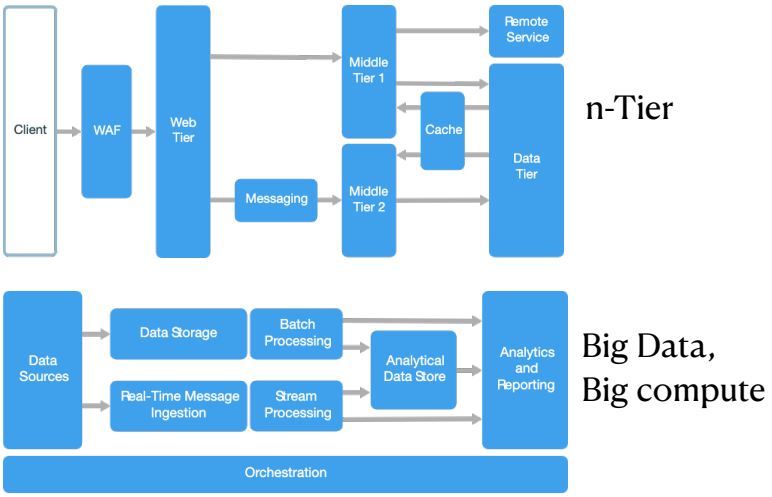
Ejemplos de Arquitecturas *N-Tier*



Ejemplos más complejos de Multi-tiered



a) JEE (Java/Oracle)



b) Microsoft Azure (Cloud)

Estilo arquitectónico N°3: Arquitectura de servicios

Arquitectura de servicios

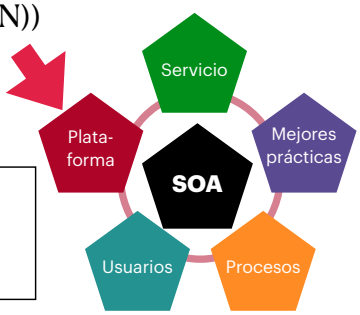
Evolución desde Cliente-Servidor

- **Arquitecturas Cliente-Servidor y Computación en Red (app. 1975- 1993)**
  - Servidores centrales (e.g. Archivos, naming, RDBMS); Arquitecturas 2-Tiers & 3-Tiers.
  - Primeros mecanismos de RPC (e.g. Sun-RPC y DCE/RPC); desarrollo de objetos distribuidos.
- **Arquitecturas basada en Objetos y Componentes Distribuidos (app. 1990-2010)**
  - *Middleware* con comunicación O-O y modularización mediante componentes administrables por contenedores.
  - Corba (OMG), Java RMI & JEE (Sun Microsystems; Oracle), DCOM & .NET (Microsoft).
- **Servicios Web y Arquitectura orientada a servicios (SOA) (app. 2000-hoy)**
  - Exposición de interfaces estándares para facilitar integración en Internet (Web).
  - Tecnologías Web (XML, SOAP, WSDL, UDDI); ESB como broker de integración.
- **Arquitectura de Microservicios (2010-hoy)**
  - REST y protocolos livianos basados en HTTP; Aplicaciones nativas Cloud & Contenedores; API gateways.
- **Arquitectura FaaS y Serverless (2014-hoy)**
  - Arquitectura dirigida por eventos en Cloud, sin administración de infraestructura y con auto-escalado (e.g. AWS Lambda, Google Cloud Functions).

a) Arquitectura orientada a servicios (SOA)

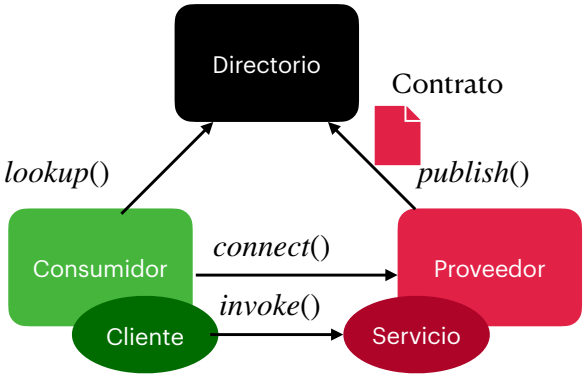
DEFINICIÓN: Estructura aplicaciones complejas como un conjunto de servicios compuestos y reutilizables. Servicios son autónomos, modulares y se comunican por una red, para implementar típicamente la "lógica de negocio".

- **Características**: Promueve acoplamiento débil, abstracción (interfaces), resusabilidad, composición e interoperabilidad. Facilita integración entre diferentes organizaciones.
- **Protocolos de comunicación**: SOAP (XML), REST (HTTP+JSON))
- **Servicios de apoyo**: *Brokers* de mensajes, descubrimiento, seguridad.



Evaluación:
+ Separación de responsabilidades (*concerns*), modularidad y reusabilidad; escalabilidad
– Sobrecarga en la gestión de servicios y la comunicación.

Componentes de SOA



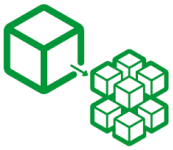
- **Servicio**: Unidad funcional autónoma (e.g., servicio de procesamiento de pagos, servicio de autenticación de usuarios).
- **Proveedor de servicios**: Entidad que implementa y presta el servicio.
- **Consumidor de servicios**: Entidad que utiliza el servicio a través de una red.
- **Contrato de servicio**: Especificación formal del servicio (e.g., interfaz, parámetros de E/S y protocolo de comunicación).
- **Registro de Servicios**: Directorio para publicar y descubrir servicios registrados (e.g., protocolo UDDI).
- **Broker de servicios (ESB)**: Infraestructura de *middleware* que facilita la comunicación entre servicios (e.g. para enrutamiento, transformación y orquestación).

b) Arquitectura de Microservicios

DEFINICIÓN: Estilo arquitectónico en el que las aplicaciones se crean como un conjunto de servicios pequeños, independientes, poco acoplado y con una gobernanza descentralizada (en contraste con SOA).

Características:

- Enfocados en una única capacidad de negocio
- Autónomo (cada servicio posee sus propios datos y lógica)
- Independencia de implementación y despliegue
- Más escalable, mejor rendimiento y mayor agilidad en el desarrollo



Ejemplo de uso: Netflix, Amazon, Uber

Evaluación:

\+ Autonomía de equipos, agilidad, mantención y despliegue CI/CD; escalabilidad, diversidad tecnológica

– Complejidad por mayor #componentes, *overhead* operacional, latencia y fiabilidad de red, consistencia de datos, *testing*.

c) Arquitectura Serverless y FaaS

DEFINICIONES:

- ***Serverless Computing* (sin servidor)**. Modelo computacional en la nube donde desarrolladores despliegan aplicaciones sin gestionar la infraestructura subyacente (i.e., automatizado por el proveedor *Cloud*).
- **FaaS (Function as a Service)**. Usa *Serverless* para desplegar funciones (en lugar de aplicaciones completas), que se ejecutan bajo demanda, normalmente en respuesta a eventos, y finalizan una vez completada la tarea.

CARACTERÍSTICAS CLAVE:

- **Sin estado y corta vida**. Cada función se llama independientemente y no persisten.
- **Dirigida por eventos**. Funciones se ejecutan como respuesta a un evento.
- **Escalabilidad y concurrencia**. Funciones se ejecutan en paralelo y servicios *Cloud* apoyan a escalar según demanda de recursos.

SERVICIOS REQUERIDOS:

- Soporte para FaaS (e.g. desde proveedores *Cloud*: AWS Lambda, Google Cloud Functions, Azure Functions)
- Manejo de mensajes y eventos (Pub/Sub colas de mensajes, eventos IoT)
- *Serverless* incluye uso de base de datos y almacenamiento de datos (e.g. AWS DynamoDB)

Estilo arquitectónico N°4: Arquitectura de mensajería

Arquitectura de mensajería

DEFINICIÓN: Estilo arquitectónico basado en sistemas de mensajería, plataformas de software que permiten el intercambio de mensajes entre aplicaciones, servicios o sistemas de forma desacoplada. Facilita comunicación asíncrona y mejora la escalabilidad y fiabilidad.

Características clave:

- **Desacoplamiento**: Productores y consumidores de mensajes operan independientemente y se comunican indirectamente a través de un intermediario con capacidad de almacenamiento.
- **Durabilidad**: Soporte para almacenar mensajes en medios persistentes (e.g., para procesamiento asincrónico y resiliente).
- **Escalabilidad**: Soporte eficiente para grandes volúmenes de mensajes y escalamiento horizontal.
- **Garantía de entrega**: *At-most-once*, *At-least-once* y *Exactly-once*.
- **Ordenamiento**: Asegura que mensajes sean procesados en la secuencia correcta.

Tipos de sistemas de mensajería (se profundizará en siguiente capítulo):

1. Sistemas orientados a mensajes (MoM: *Message oriented Middleware*)
2. Sistemas dirigidos por evento (para arquitecturas dirigidas por eventos)
3. Sistemas de mensajes transientes (no durables; más livianos y adecuado para ciertas aplicaciones)

a) Arquitectura orientada a mensajes (AoM)

También: MoM (*Message-oriented-Middleware*)

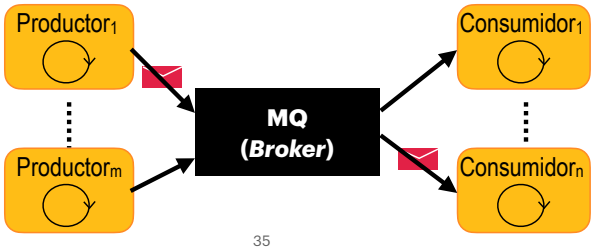
DEFINICIÓN: Estilo arquitectónico donde componentes se comunican indirectamente, intercambiando mensajes a través de un conector intermediario (e.g. *broker* o cola de mensajes) con capacidad de almacenamiento persistente. Mensajes típicamente representan peticiones o comandos.

Conceptos clave:

- Comunicación asíncrona y desacoplada a través de un Message Broker (e.g., uso de servicios tales como RabbitMQ, Kafka, AWS SQS, Apache ActiveMQ, etc.)
- Patrones de comunicación: Point-to-Point (*one-to-one*) vs. Publish-Subscribe (*one-to-many*); a veces también *Request-Reply*.

API Estándares:

- JMS (Java)
- AMQP (OASIS)
- MQTT (Telemetría)



b) Arquitectura dirigida por eventos (AdE)

DEFINICIÓN: Estilo arquitectónico donde los eventos dirigen el flujo de datos y gatillan reacciones en diferentes servicios para procesarlos. El productor genera eventos, que representan cambios de estado y el consumidor los procesa.

Características clave:

- Comunicación asíncrona y desacoplada.
- Procesamiento en tiempo real (e.g. restricciones de latencia y procesamiento).
- Consistencia eventual (procesamiento asincrónico puede producir inconsistencias por un lapso de tiempo, pero estado converge).
- Escalabilidad y tolerancia a fallos.

Estilo arquitectónico Nº5: Arquitectura de datos compartidos

Arquitectura de Datos compartidos

DEFINICIÓN: Estilo arquitectónico donde múltiples procesos de un sistema interactúan indirectamente a través de un espacio de datos compartidos (tb. memoria o repositorio de datos). Los datos pueden estar o no estructurados y pueden ser volátiles, temporales o persistentes.

Características clave:

- Desacoplamiento de componentes a través de una capa de datos común.
- Requieren abordar el problema de sincronización y consistencia de datos (por acceso concurrente).
- Durabilidad de datos (con o sin persistencia).

Ejemplos:

- Memoria compartida distribuida (e.g. Ivy: volátil)
- Espacios compartidos (e.g. Espacio de tuplas como Linda, JavaSpaces, Zookeeper)
- Almacenes de archivos u objetos (e.g. NFS, HDFS, Amazon S3)
- Sistemas de gestión de datos (e.g. DBMS, BigQuery, NoSQL)

a) Memoria compartida y Espacio compartido

Memoria compartida distribuida (DSM). Abstracción que permite a varios procesos en diferentes nodos de un sistema distribuido compartir memoria (volátil), como si se tratara de un espacio de direcciones unificado, aunque la memoria física esté distribuida.

- Modelo: Procesos de diferentes máquinas pueden leer y escribir en la memoria (sin pasar mensajes explícitos), simplificando la programación paralela y facilitando portabilidad.
- Ejemplos: Treadmarks; Kernel based VM (KVM) comparte páginas entre VM.

Espacio de tuplas. Estilo arquitectónico basado en la coordinación de procesos, que interactúan indirectamente a través de un espacio de datos compartido y estructurado (volátil).

- Modelo: Basado en paradigma del espacio de tuplas inmutables, donde los procesos leen, escriben (insertan) y remueven tuplas de datos de un espacio común.
- Ejemplos: JavaSpaces (volátil); Zookeeper (con registros persistentes tipo clave-valor).

Fuentes:
- B. D. Fleisch. 1987. "Distributed shared memory in a loosely coupled distributed system." SIGCOMM 17, 5 (Oct./Nov. 87), 317–327.
- David Gelernter. 1985. "Generative communication in Linda." ACM ToPLaS 7, 1 (Jan. 85), 80–112.

b) Sistemas de almacenamiento de datos persistentes

DEFINICIÓN. Estilo arquitectónico en que múltiples procesos o componentes independientes de un sistema interactúan asincrónica e indirectamente a través de un espacio compartido de datos almacenados en un medio persistente.

Características clave:

- **Estructura de los datos** (no estructurados, semi-estructurados y estructurados)
- **Durabilidad de datos** (asegurar durabilidad en medio persistente)
- **Lenguaje de consulta** (e.g. SQL, búsqueda clave-valor, SPARQL para grafos)
- **Sincronización** (asegurar consistencia en accesos concurrentes a los datos)
- **Escalabilidad y tolerancia a fallos**. El sistema soporta cargas crecientes y fallas, introduciendo redundancia o más recursos.

Taxonomía: Existe una gran variedad de estos sistemas, con diferentes propósitos y propiedades, como:

- Sistemas de archivos distribuidos, almacenes de objetos, Sistemas de base de datos (relacional, analítica de datos, NoSQL, etc.), Coordinación distribuida (e.g. Zookeeper).

Ejemplos: Sistemas de almacenamiento de datos persistentes

1. **Almacenes de datos** (datos brutos, no estructurados)
   - **Sistemas de archivos**: archivos organizados jerárquicamente (e.g. NFS, HDFS yAmazon EFS)
   - **Almacenes de objetos**: datos no estructurados (e.g., Amazon S3, Google Cloud Storage, Azure Blob Storage)
2. **Bases de datos NoSQL** (repositorios datos, no relacionales y optimizadas para escalar)
   - **Clave-valor**: estructura y operadores simples (e.g. Amazon DynamoDB, Redis in-memory).
   - **Columnares**: tabla de muchas columnas (e.g., Google Bigtable, Apache HBase, Apache Cassandra).
   - **Documentos**: semi-estructurado en formato XML o JSON (e.g., MongoDB, Apache CouchDB).
   - **Grafos**: nodos y arcos/relaciones (e.g., SPARQL, Azure Cosmos DB).
3. **Repositorios de datos** (datos estructurados con metadatos, indexación y lenguajes de consulta):
   - **BD transaccionales**: tipo RDBMS o OLTP (e.g., PostgresDB, MySQL).
   - **BD analíticas para BI**: tipo OLAP, DWH (e.g., Google BigQuery, Amazon Redshift)
   - **Data Lakes**: almacenamiento masivo de datos brutos estructurados y no estructurados
4. **Coordinación distribuida**: elección de líder, configuración y descubrimiento (e.g.,Apache Zookeeper, etcd, Consul).

Caching Distribuido

Integración de *caching* en Repositorios de datos distribuidos

- **Características**
  - **Mejora de rendimiento**. Reduce acceso lento a sistemas de almacenamiento de datos.
  - **Escalabilidad**. Servicio de *caching* puede escalar horizontalmente.
  - **Alta disponibilidad y tolerancia a fallos**. Se replica la función y diferentes nodos pueden asumir la función de *caching* en caso de falla.
- **Casos de uso**
  - **Aplicaciones Web** (e.g. páginas HTML, datos de sesiones)
  - **Almacenes de clave-valor** (e.g., *caching* en memoria volátil: Redis y Memcached)
  - **Consultas SQL** (e.g. consultas complejas y frecuentes con datos que cambian poco)

Estilo arquitectónico N°6: Arquitectura de Redes P2P

Arquitectura Peer-to-Peer (P2P)

DEFINICIÓN: Un estilo arquitectónico con un modelo descentralizado, en el que cada nodo o par (*peer*) actúa como cliente y/o servidor. Usa una red sobrepuesta (*overlay*) que requiere de mecanismos de descubrimiento de recursos y algoritmos de ruteo.

Características clave:

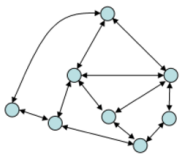
- No hay autoridad central
- Los pares se comunican directamente
- Auto-organizada, escalable y resiliente (altamente redundante)

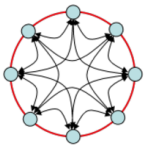
Ejemplos:

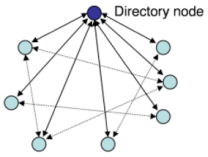
- Sistemas de intercambio de archivos (e.g. BitTorrent, Gnutella)
- Redes Blockchain (e.g. Bitcoin, Ethereum)
- Redes de distribución de contenidos (e.g. WebTorrent, eCDN de MS)

Tipos de redes P2P (según estructura)

- **Redes P2P no estructurados**: Los pares se conectan aleatoriamente, sin existir una topología específica.
  - Ejemplo: Gnutella, los primeros Napster.
- **Redes P2P estructurado**: Los pares se organizan en una topología específica. Típicamente usan DHT (*Distributed Hash Table*) para una búsqueda eficiente de recursos.
  - Ejemplo: Chord, Kademlia.
- **P2P híbrido**: Combina P2P con componentes centralizados. Algunos nodos (supernodos) actúan como servidores para un subconjuntos.
  - Ejemplo: Skype, BitTorrent (con *trackers* o rastradeares).







Redes P2P

Ventajas y desafíos

Ventajas:

- Descentralización.
- Escalabilidad y distribución de carga.
- Tolerancia a fallos (no hay un único punto de fallo y existe redundancia de contenidos).
- Utilización eficiente de los recursos.

Desafíos:

- Descubrimiento de nodos y gestión de *peers*.
- Riesgos de seguridad (e.g. DoS y ataques Sybil).
- Coherencia de datos y calidad de contenidos.
- Alta tasa de rotación de nodos.

Lectura complementaria: van Steen (2023). Sección 2.4 "*Symmetrically distributed system architectures*", pp. 88-98.

2.3 Nombramiento (*Naming*) y descubrimiento

2.3.1 Concepto de *Naming*

Motivación a nombramiento (*naming*)

Concepto de nombre y ligado de recursos

- **Nombre y ligado (*binding*) en computación:**
  - En **Lenguajes de programación** (LP) y **Sistemas operativos** (SO) y, un nombre se usa para identificar y/o referenciar en un programa algún objeto o recurso del sistema (entidad).
  - La resolución del nombre típicamente liga (*bind/link*) a una entidad una dirección en memoria.
  - Nombre permite acceder y compartir recursos. Por ejemplo:
    - **Programas**: Nombre → Dirección de memoria (e.g. objeto de datos, variable o subprogramas).
    - **Memoria virtual**: Dirección lógica → Dirección física (quizás cargando página desde memoria secundaria)
    - **Archivos:** Nombre (*string*) → Descriptor (proceso) → Índice global (*kernel*) →*inode* → # bloque en disco.
- **En redes de computadores se distinguen tres tipos de referencias** (Shoch, 1978):
  - **Nombre**: identifica un recurso; es decir, qué se quiere buscar.
  - **Dirección** (*address*): indica dónde está el recurso.
  - **Ruta** (*route*): nos dice cómo llegar a ese recurso.

Fuente: Shoch, J. (1978). Inter-network naming, addressing, and routing. In *Proc. of IEEE Computer Conference, COMPCON* (Washington, D.C., Fall). IEEE, New York, pp. 72-79.

Nombramiento en Sistemas distribuidos

DEFINICIÓN: En redes y sistemas distribuidos, *naming* (nombramiento) se refiere al proceso de asignar y resolver nombres o identificadores de recursos (e.g. servidores, objetos, archivos y servicios remotos).

PROPÓSITO: En una arquitectura de un sistema distribuido, permite descubrir y acceder a recursos (remotos), sin requerir conocimiento (previo) sobre su ubicación física.

- Resolución de nombres es esencialmente dinámico.
- Apoyo a la transparencia de ubicación, reubicación, migración, replicación y fallas, entre otras.
- Servicio de nombramiento es requerido para descubrir dinámicamente componentes, siendo servicio crítico que requiere alta disponibilidad, resiliencia y seguridad.

Definiciones básicas

- **Entidad**. Corresponde a los componentes, objetos o recursos de un sistema distribuido.
  - Ejemplos: usuario, servicio, nodo, puerto, proceso, función o objeto de programa, etc.
- **Nombre**. Una secuencia de *bits* o caracteres que permite referenciar una entidad en un sistema distribuido, con el propósito de comunicarse con ella o de compartirla con otros.
- **Espacio de nombres** (*name space*). Conjunto de nombres válidos que cumplen una cierta sintaxis o convención.
  - Nombres son normalmente asignados por una autoridad administrativa.
  - Espacio se puede segmentar para descentralizar y delegar administración de un subespacio (escalar mejor.)
- **Ligado de nombre** (*name binding*). Proceso de asociar el nombre de una entidad con un recurso específico.
  - Puede ser estático (e.g. IP fija, servicio de red) o dinámico (e.g. DHCP, microservicio).
  - Introducir una indirección puede ser útil para mayor transparencia y flexibilidad.
    - Ejemplo: Se nombra un servicio ofrecido por un *cluster* de servidores, para transparencia de replicación.
- **Nombramiento** (*naming*). Para entidades, mapeo entre nombres, identificadores, direcciones u otros atributos.

Tiempo de ligado (*binding*)

Algunas reflexiones sobre su relación con Sistemas distribuidos

- **Tiempo de compilación** (estático o temprano). Direcciones fijas son de uso eficiente, pero inflexibles y dificultan compartir (dinámicamente).
  - Ejemplo en SD: Servicios estándares tienen direcciones estáticas (e.g. SSH, FTP, SMTP, HTTP).
- **Tiempo de carga**. Direcciones se resuelven al momento de cargar el programa en memoria; más flexible para reubicar y compartir (e.g. variables globales, bibliotecas compartidas — *shared libraries*).
  - Ejemplo en SD: Mejorar gestión de memoria en servidores (e.g. activación y pasivación de objetos)
- **Tiempo de ejecución** (dinámico o tardío). Se resuelve en tiempo de ejecución (e.g. objetos de *stack y heap*). También acceso con carga dinámica de objetos a la memoria (e.g. DLL).
  - Ejemplo en SD: Resolver tardíamente reduce *overhead* de resolución y apoyo adaptación del sistema a condiciones operativas.
    - "Lo que no se usa no se resuelve".
    - Escala mejor y facilita reconfiguración dinámica (e.g. agregar recursos, balancear carga, recuperarse de errores, etc.).

Clasificaciones de nombres

- **Nombre orientado a humanos vs. orientado a sistemas**:
  - **Nombre orientado a humanos**: Es un nombre de caracteres legible o amigable
    - Este tipo de referencia se denomina a veces simplemente "nombre"
    - e.g. nombre de dominio: `www.inf.utfsm.cl` ; o nombre de archivo: `docs/file.pdf`
  - **Nombre orientado a sistemas**: Para un uso eficiente por los sistemas y/o máquinas
    - e.g. `200.1.19.23` o `0x34:a5:f2:37:54:c3`
- **Nombre plano vs. estructurado**:
  - **Nombre plano**: El nombre no tiene estructura (e.g. `Alicia` o `21423`).
  - **Nombre estructurado**: El nombre está estructurado en varios campos (e.g. `200.1.19.23`).
    - Nombre contiene información tal como ubicación u otros atributos.
    - Pueden ser un nombre compuesto o jerárquico (e.g. `urn:isbn:0451450523`; o `/usr/remote/bin/ssh`).

Unicidad de nombres (UID)

Es fácil lograrlo en espacios de nombre pequeños, pero complicado a gran escala.

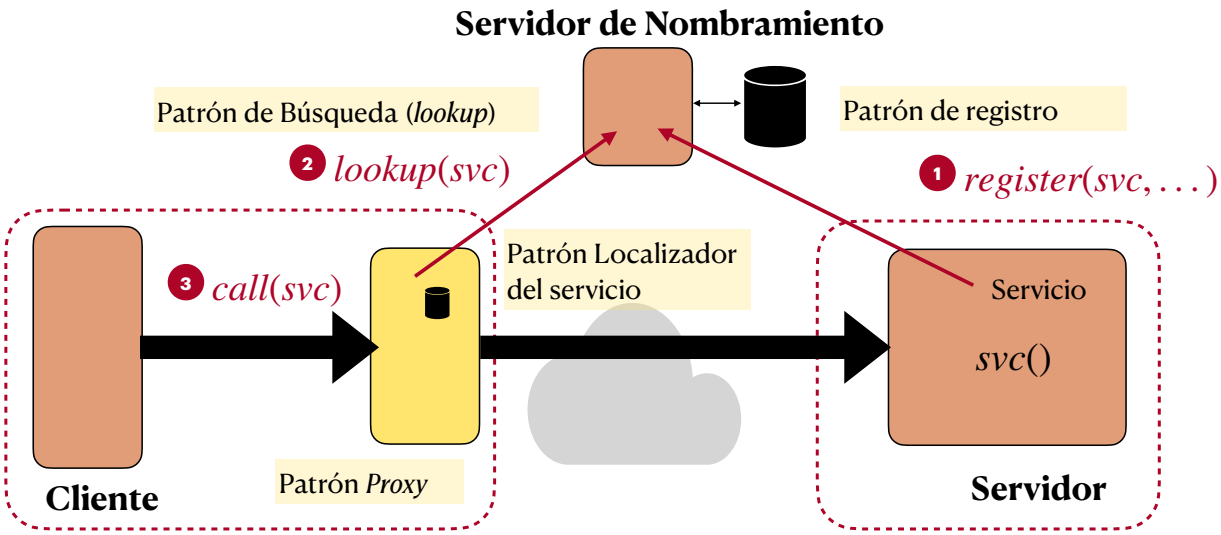
- **Unicidad para nombres planos (orientado al sistema)**. Se establece un prefijo para segmentar el espacio de nombre (luego, pasa a ser estructurado). Ejemplos:
  - Dirección IPv4 de 32 bits: $< red, host >$
  - Dirección Ethernet de 6 bytes: *<3 bytes para fabricante, 3 bytes para unidad controladora>*
  - PID global: $< IP_{nodo}, PID_{local} >$
- **Unicidad para nombres estructurados (orientado al humano)**. Nombre compuesto define una jerarquía mediante varios nombres conectados por separadores. Ejemplo:
  - Nombre de archivo: $/usr/local/anamaria/docs/tarea1.docx$
  - URL: $rmi://vm1.inf.utfsm.cl:1099/aplicacion/calculadora$

OBSERVACIÓN: Nombres a veces son sólo únicos en un determinado contexto, pero no globalmente.

Modelos de nombramiento

- **Nombramiento plano**
  - Sin estructura; nombres son identificadores únicos (e.g., UUID/GUID, direcciones MAC).
  - Difícil de gestionar a gran escala, pero eficaz en la búsqueda.
- **Nombramiento jerárquico**
  - Estructurado como un árbol (e.g., nombre en DNS; ruta o *pathname* de archivos).
  - Se adapta bien, pero requiere más pasos de búsqueda para resolver.
- **Nombramiento basado en atributos**
  - Identifica los recursos por propiedades en vez de nombres, tipo "páginas amarillas" (e.g., consultas a directorio LDAP: "*encontrar impresora laser cerca de mi oficina*").
  - Es flexible, pero requiere indexación y un procesamiento eficaz para las consultas.

Ejemplo de patrones



2.3.2 Resolución de nombres

Resolución de nombres

DEFINICIÓN: Corresponde al proceso de mapear un nombre de una entidad a alguno(s) de sus atributos (usualmente la dirección).

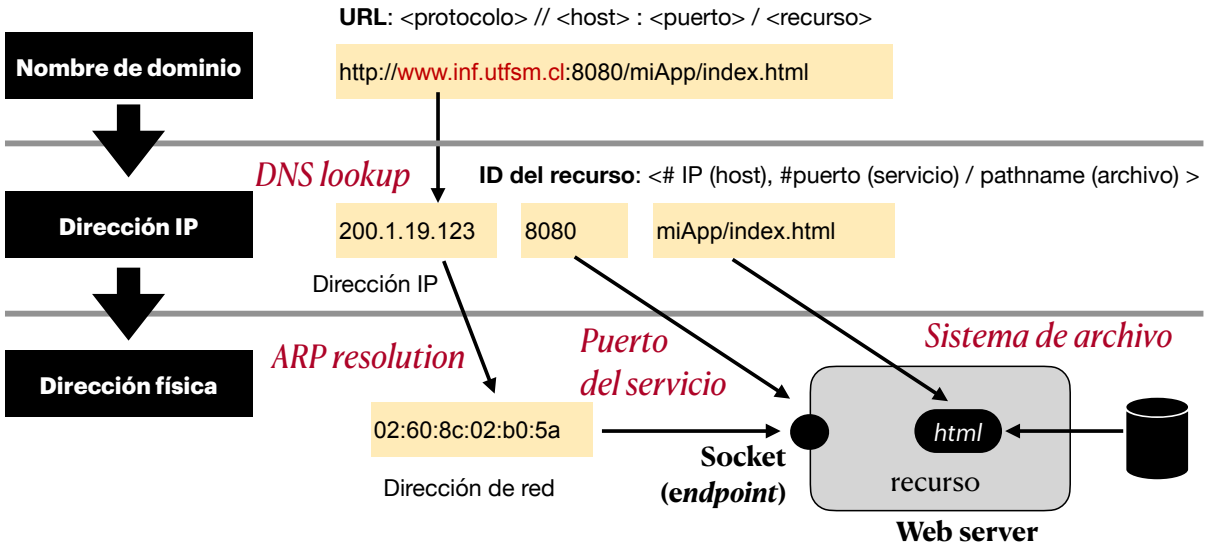
Características:

- Se debe buscar y encontrar en el espacio de nombres cierta información asociada a una entidad.
- Proceso de búsqueda puede ser simple (e.g. *broadcasting*) o repetitivo (e.g. iterativo o recursivo).
- Ocurre a un cierto nivel de abstracción (*layer)*, pero su implementación puede requerir bajar múltiples niveles (e.g. recursivo) para resolver finalmente.

Mecanismo de clausura:

- Si la información está distribuida y/o particionada en múltiples sitios, se debe saber dónde comenzar la búsqueda (partida) y asegurar que finalmente se encuentre lo buscado (término).
- Una búsqueda repetitiva en diferentes sitios requiere aproximarse sucesivamente a aquel sitio que tenga la información buscada para resolver (convergencia).

Ejemplo de resolución de nombre



Fuente: adaptada de Coulouris, et al. (2012), pag. 567

a) Resolución de nombres planos

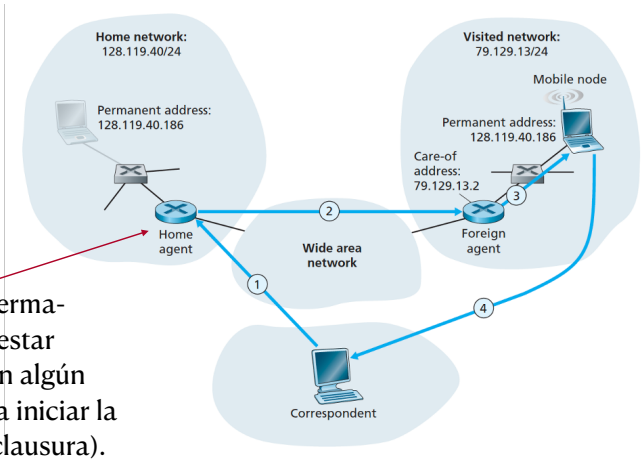
1. **Difusión** (*broadcasting*). *Broadcast* de un identificador de una entidad en una red, respondiendo la entidad referenciada con su dirección (e.g. ARP). Normalmente se realiza *caching* de direcciones resueltas.
   - No escala bien más allá que una red de difusión (e.g. LAN).
   - Requiere que todos los procesos estén escuchando solicitudes de resolución (mayor *overhead*).
2. **Punteros de avance** (*forwarding pointers*). Cuando una entidad se mueve, deja detrás un puntero hacia la siguiente ubicación (pista).
   - Seguimiento de direcciones (o pistas) para resolver un nombre es transparente para el cliente.
   - No escala bien si cadenas son muy largas o con grandes distancias (alta latencia y overhead).
   - *Caching* puede mitigar el problema.
3. **Enfoque de base o domicilio** (*home based*). Cada entidad tiene una dirección permanente en un domicilio (*home base*).
   - Se mantiene en el domicilio registro de la dirección de ubicación actual.
   - Solución estándar en redes móviles.

OBSERVACIÓN: Los 3 mecanismos soportan bien cambios de configuración y movilidad.

Consulta sobre otros mecanismos:
van Steen, et al. (2023), cap. VI: ODHT en redes P2P (anillo) y enfoque jerárquico.

Soporte para movilidad IP

Ejemplo basado en domicilio (*home-based*)



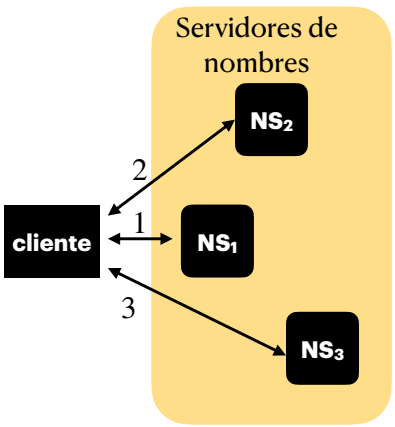
Dirección permanente debe estar registrada en algún servicio para iniciar la búsqueda (clausura).

- **Home location**: rastrea ubicación actual de una entidad móvil.
- **Home agent**: contacto inicial para resolver.
- **Care-of-address**: dirección temporal en otra red, que debe registrar el home-agent.

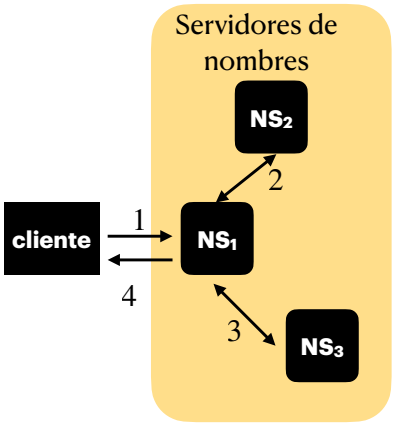
Fuente: Jim Kurose, Keith Ross, "*Computer Networking: A Top Down Approach*", 6th edition, Addison-Wesley, 2012.

b) Resolución de nombres estructurados

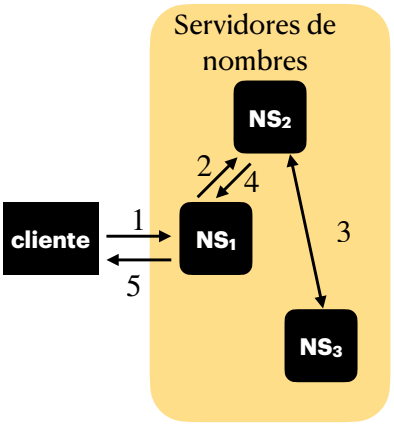
Métodos con información distribuida



a) Iterativo controlado por el cliente

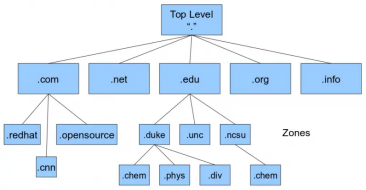


b) Iterativo controlado por el servidor



c) Recursivo controlado por el servidor

2.3.3 Nombramiento y descubrimiento de componentes



Servicios de Nombramiento: nombres estructurados

Servicios de Nombre vs. Servicios de Directorio

Servicio de nombre:

- Establece convención y sintaxis para nombres estructurados de entidades.
- Nombres definidos en un contexto son únicos para cada entidad.
  - Multiplicidad ocurre por nombramiento especial (e.g. enlace simbólico o alias).
- Puede conectar un conjunto de contextos del mismo tipo (misma convención de nombres), distribuidos en múltiples servidores.
- Ejemplos: Sistemas de archivos distribuidos (e.g. Sun NFS o AFS) y DNS (TCP/IP: RFC-1035).

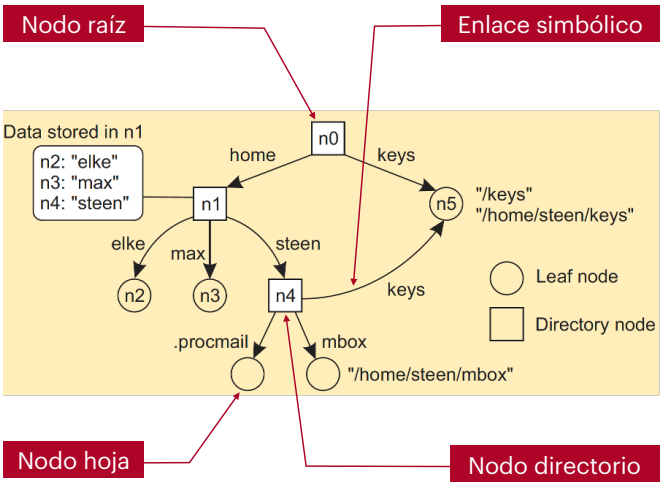
Servicio de directorio:

- Extensión de un Servicio de Nombre:
  - Permite que entidades tengan varios atributos, por lo que se les caracteriza como nombramiento basado en atributos.
- Permite realizar búsquedas basadas en atributos (tipo páginas amarillas).
  - e.g. “encontrar impresora laser en el edificio F con velocidad $\geq$ 60 [pag/s]”.
- Ejemplos: X.500 (ISO/OSI + ITU-T), LDAP (TCP/IP: RFC-4511) y RDF (W3C).

OBSERVACIÓN: “Servicios de registro” de alta disponibilidad y fiabilidad se revisarán en cap. VI

Espacio de nombres (Directorio)

- **Nombres** compuestos (estructurados), definiendo ***path names*** (camino o sendero) en un **grafo dirigido**.
  - **arcos** se etiquetan con un nombre.
  - Existe un único **nodo raíz**.
  - Cada **nodo** puede contener múltiples atributos (e.g. tipo, ID, dirección, alias, dueño, permisos, etc.).
- **Nodos hoja** (*leaf*) corresponden a entidades (nombradas).
  - Atributos del nodo definen a la entidad.
- **Nodos directorio** (intermedios) son entidades que se refieren a otros nodos.
  - Contiene pares <ID nodo, nombre arista>, posiblemente con varios atributos.
- **Enlace simbólico** se define agregando en un nodo directorio un nombre absoluto (referencia especial).
  - Arco cruzado crea otro nombre para una entidad.
  - Puede generar ciclos al referirse a nodos directorio.



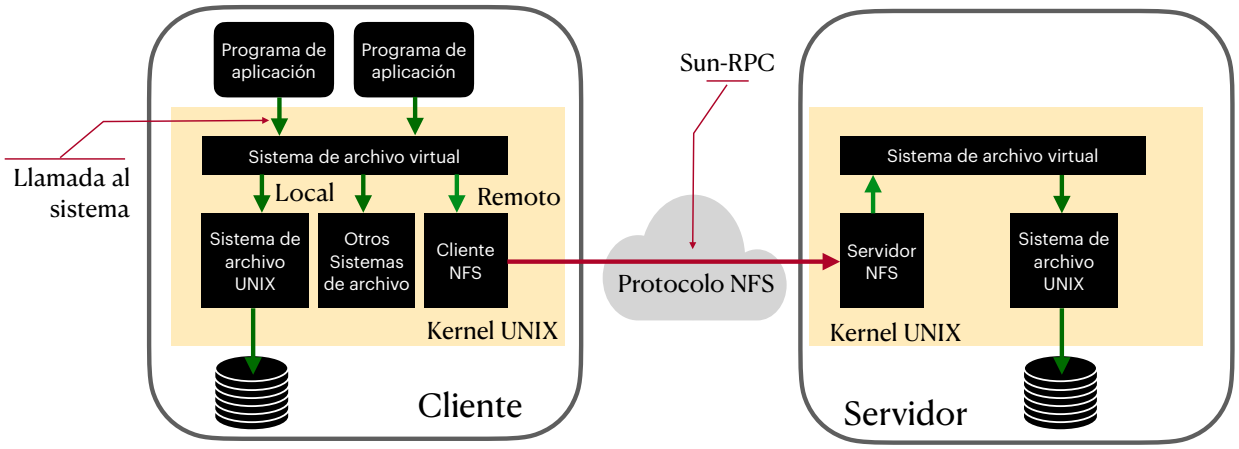
Fuente: van Steen, et al. (2023)

Metas de diseño para Servicios de Nombramiento

- **Escalabilidad y alto desempeño.** Los sistemas de nombres deben soportar millones de búsquedas de manera eficiente.
  - Particionamiento (*sharing*) en múltiples dominios administrativos y servidores para descentralizar mantención de datos y resolución de nombres.
  - Uso de *caching* para nombres resueltos y replicación para distribuir carga.
  - Se acepta consistencia eventual para mejor desempeño y escalabilidad.
- **Alta disponibilidad y resiliencia**. Se replica los datos para mayor disponibilidad, pero se debe gestionar fallos y recuperación de errores (e.g. redundancia de DNS).
- **Seguridad**. Prevención de suplantación de identidad, fuga de datos sensibles, integridad de datos almacenados, envenenamiento de *cache* DNS, búsquedas no autorizadas, etc.

Caso Nº1: Sistema de archivo Sun-NFS

Arquitectura de Sun-NFS



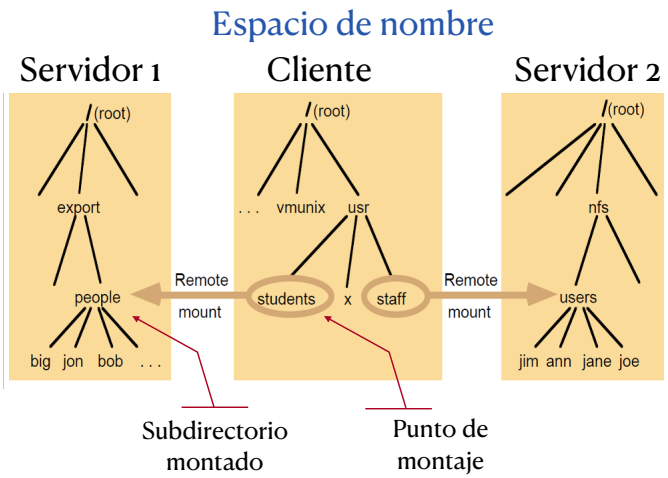
Fuente: Coulouris, et al. (2012)

- **Escalamiento**: Clientes pueden compartir y distribuir espacio de almacenamiento y carga entre varios servidores.
- **Transparencia de acceso y ubicación**: Archivos remotos y locales se acceden de igual manera por el cliente.

Sun-NFS: Montaje de volúmenes remotos

Características:

- Soporta montar (*mount*) subárboles del espacio de nombres de otro servidor (remoto) en el espacio de nombres propio (local).
- Permite replicar en modo lectura (e.g. binarios), para mayor disponibilidad y distribuir carga entre servidores.



Fuente: Coulouris, et al. (2012)

Equivalencia de nombres:

- `/usr/students` montado en el **Cliente** corresponde a: `/export/people` en **Servidor 1**.
- `/usr/staff` en montado en el **Cliente** corresponde a: `/nfs/users` en **Servidor 2**.

Caso Nº2: LDAP (Lightweight Directory Access Protocol)

Servicio de directorio, estándar RFC 4511

- Define una Base de Información de Directorio (DIB), especializada para proveer un servicio de directorio, que se accede mediante protocolos de TCP/IP.
- Basado en X.500, espacio de nombres se encuentra organizada jerárquicamente en un árbol.
  - Permite establecer fronteras geográficas y organizacionales (u otros modelos).
  - Cada entrada del directorio (registro) es un nodo del árbol, tiene nombre único y contiene pares `<atributo, valor(es)>`.

Attribute	Abbr.	Value
Country	*C*	NL
Locality	*L*	Amsterdam
Organization	*O*	VU University
OrganizationalUnit	*OU*	Computer Science
CommonName	*CN*	Main server
Mail_Servers	–	137.37.20.3, 130.37.24.6, 137.37.20.10
FTP_Server	–	130.37.20.20
WWW_Server	–	130.37.20.20

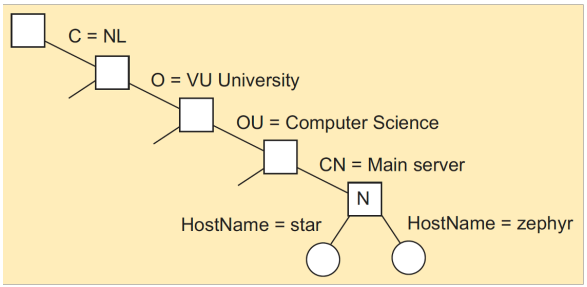
Observaciones:

- Usado ampliamente en **ambientes UNIX** y en redes interorganizacionales.
- **Active Directory** de Microsoft es compatible con LDAP, lo que facilita integración con ambientes UNIX.

Fuente: van Steen, et al. (2023)

Ejemplo de LDAP

Corresponde a un repositorio de datos



Attribute	Value
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Comp. Sc.
CommonName	Main server
Host_Name	star
Host_Address	192.31.231.42

Attribute	Value
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Comp. Sc.
CommonName	Main server
Host_Name	zephyr
Host_Address	137.37.20.10

Observaciones:

- Vista parcial del árbol del Directorio.
- Se define nombre único global; e.g., `/C=NL /O=VU Univesity /OU=Computer Science` (Similar a nombre de dominio `cs.vu.nl` en Internet).
- Se definen dos pares en la entrada de directorio "CN= main server"
- Existe definido un lenguaje de consultas para obtener diversa información del Directorio.

Fuente: van Steen, et al. (2023)

2.4 Conclusiones

Lecciones aprendidas

- Se han reconocido los principales **estilos arquitectónicos** en Sistemas distribuidos y algunos patrones arquitectónicos o de diseño asociados. Se ha puesto énfasis en **estilos de interacción** entre componentes. Se destaca los siguientes estilos:
  - Estilos de estratificación: **Multi-tiered** y **Multi-layered**.
  - Estilos de servicios: **cliente-servidor**, **objetos y componentes distribuidos**, **arquitectura orientada a servicios** (SOA) y **microservicios**.
  - Estilos indirectos de **mensajería** y **dirigidos por eventos**.
  - Finalmente, estilos de **datos compartidos** y **P2P**.
- El nombramiento y servicios para **descubrimiento y localización de componentes** de una arquitectura distribuida, son necesarios para apoyar la reconfiguración dinámica de los cambios en una arquitectura de un sistema que evoluciona y se adapta a condiciones operativas.
  - Se revisaron métodos de resolución de nombres planos (difusión, punteros de avance y enfoque de base) y estructurados (servicios de nombre y directorio).

Material de estudio complementario del capítulo

- van Steen (2023). Texto guía.
  - Cap II: Architecture
    - Sección 2.1-2.4: pp .55-98 (hasta “Symetrically distribuye systems (P2P)”)
  - Cap. V: Naming
    - Sección 6.1-6.2: pp. 325-332 (hasta 6.2.2 “Home-based approaches”)
    - Sección 6.3-63.4: pp. 344-379 (hasta 6.4.2 “Hierarchical implementatiosn: LDAP)
- Mark Richards & Neal Ford (2020). *Fundamentals of Software Architecture*. O’Reilly.

Capítulo II: Arquitectura en Sistemas distribuidos

Estilos arquitectónicos, patrones y sistemas

Prof. Dr.-Ing. Raúl Monge Anwandter ♦ 2º semestre 2025