Introducción a la Ingeniería de Software

Arquitectura de Software

Bibliografía

- Software Engineering 7ed
  Addison Wesley
  Ian Sommerville
- Documenting Software Architectures – Views and Beyond
  Addison-Wesley
  Paul Clements et al.
- Software Systems Architecture – Working with stakeholders using viewpoints and perspectives
  Addison-Wesley
  Nick Rozanski y Eoin Woods

Arquitectura de Software

Especificación de Requerimientos del sistema (SRS)

En este camino hay mucho por hacer.

¿Comenzamos a programar para terminar lo antes posible?

- ¿Cuáles serían los riesgos?

Sistema instalado y funcionando

Arquitectura de Software

Especificación de Requerimientos del sistema (SRS)

- Arquitectura de Software
- Diseño detallado
- Implementación
- Verificación

No es un proceso en cascada. No se está definiendo un proceso.

Sistema instalado y funcionando

Arquitectura de Software

Los sistemas complejos están compuestos de subsistemas que interactúan bajo el control de un diseño de sistema

Arquitectura de Software

- Los subsistemas que componen el sistema,
- las interfaces y
- las reglas de interacción entre ellos.

Definición

A software architecture for a system is the structure or structures of the system, which consist of elements, their externally visible properties, and the relationships among them.

Documenting software architectures, views and beyond

Importancia

Ventajas de diseñar y documentar explícitamente una arquitectura de software:

- Comunicación entre stakeholders
- Decisiones tempranas de diseño
- Reuso a gran escala

Bass, et al, Software Architecture in Practice 2ed. Addison-Wesley.

¿Qué Afecta y qué la Determina?

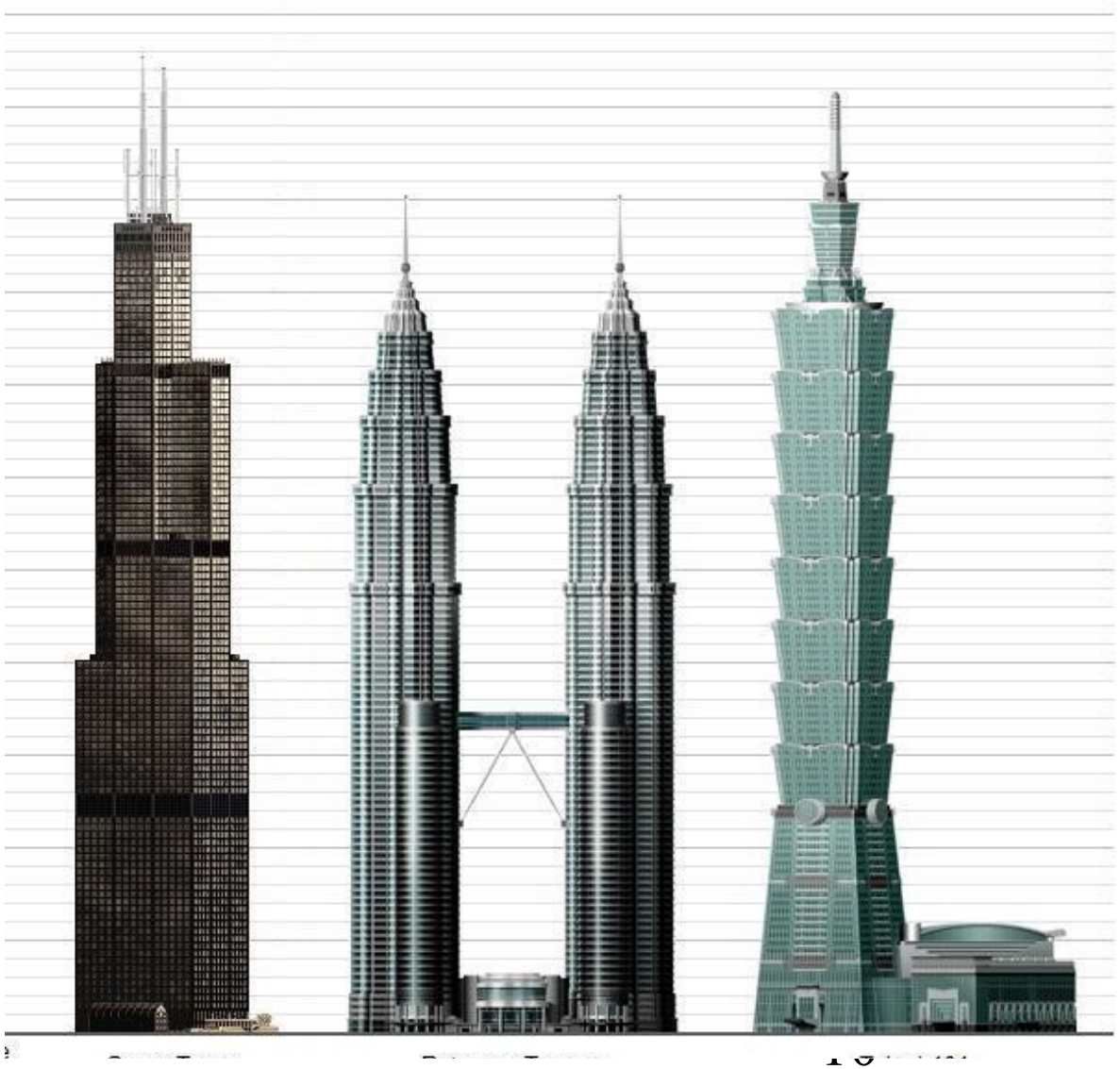
- La arquitectura de software afecta la
  - Performance
  - Seguridad (security y safety)
  - Disponibilidad
  - Mantenibilidad
  - …
- Entonces, el estilo y estructura particular elegido para una aplicación dependen fuertemente de los requerimientos no funcionales.

Conflictos entre Soluciones

- El sistema debe ser “muy” performante y “muy” mantenible
- ¿Cuál es el conflicto al momento de elegir el estilo arquitectónico?
- ¿Cómo se puede solucionar?
  - Solución de compromiso
  - Diferentes estilos para distintas partes del sistema

¿Qué tan Fácil es Modificarla?

- Sears
  EEUU
  527 metros
- Petronas
  Malasia
  452 metros
- Taipei 101
  China
  508 metros



¿Qué tan Fácil es Modificarla?

Me gustaría que el ascensor quedara del otro lado

Estaría **bárbaro** que el puente estuviera 23 pisos más arriba, la vista sería mejor

¿Qué tan Fácil es Modificarla?

Burj Dubai, otros metros más arriba, Emiratos Árabes

Aún más Complicado

Patrones de Software

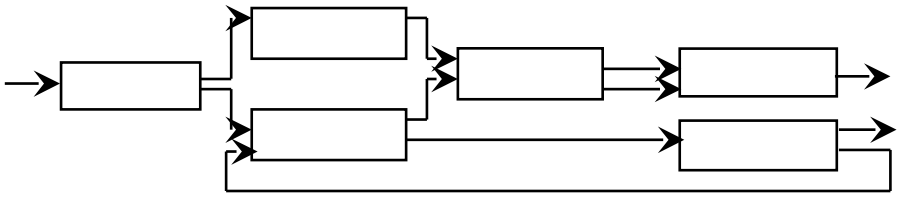
- Propósito
  - Compartir una solución probada,
  - ampliamente aplicable
  - a un problema particular de diseño.
  - El patrón se presenta en una forma estándar que permite que sea fácilmente reutilizado.
- Cinco piezas importantes de un patrón
  - Nombre
  - Contexto
  - Problema
  - Solución
  - Consecuencias (positivas y negativas)

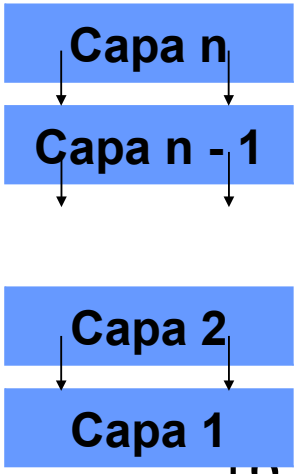
Estilos, Patrones e Idioms

- Los patrones de diseño se agrupan en tres tipos
  - **Estilos arquitectónicos:** Soluciones de organización a nivel del sistema
  - **Patrones de diseño:** Soluciones a problemas detallados de diseño de software
  - **Idioms:** Soluciones útiles para problemas específicos en algún lenguaje de programación

Estilos Arquitectónicos

- Un **estilo arquitectónico**
  - expresa un esquema de organización estructural para sistemas de software.
  - Provee un conjunto de tipos de elementos predefinidos,
  - especifica sus responsabilidades e
  - incluye reglas y guías para organizar las relaciones entre ellos





Patrones de Diseño

- Un **patrón de diseño**
  - provee un esquema para refinar los elementos de un sistema de software o las relaciones entre ellos.
  - Describe una estructura recurrente de elementos de diseño interconectados que soluciona un problema general de diseño dentro de un contexto particular

- Mencionen algún patrón de diseño que conozcan

Idioms

- Un **idiom**
  - es un patrón de bajo nivel, específico para un lenguaje de programación.
  - Describe como implementar aspectos particulares de elementos o de las relaciones entre ellos usando las características de un lenguaje particular.

Arquitectura de Software

Estilos Arquitectónicos

Beneficios de Usar Estilos

- Permite seleccionar una solución entendible y probada a ciertos problemas, definiendo los principios organizativos del sistema

- Al basar la arquitectura en estilos que son conocidos las personas entienden más fácilmente las características importantes de la misma

Formas de Uso de los Estilos

- Solución para el diseño del sistema
  - Algún estilo sirve.
  - Se adopta y se usa como una de las estructuras centrales de la arquitectura.
- Base para una adaptación
  - Algún estilo soluciona parcialmente los problemas.
  - Se puede buscar adaptar el estilo para las restricciones particulares que se tienen.

Formas de Uso de los Estilos (2)

- Inspiración para una solución relacionada
  - Ningún estilo sirve.
  - Sin embargo, entender problemas que solucionan ciertos estilos puede llevar a entender mejor el problema actual.
  - Entonces, se puede encontrar una solución que de alguna forma está relacionada con estilos existentes
- Motivación para un nuevo estilo
  - El problema no es atacado por ningún estilo encontrado
  - Es una buena oportunidad para encontrar una solución general al problema y crear un nuevo estilo

Algunos Estilos

- Shared Data (Datos Compartidos)
  - Pizarrón (Blackboard)
- Cliente-Servidor
- Capas Jerárquicas
- Descomposición Orientada a Objetos
- Tubos y Filtros
- Control Centralizado
- Control Basado en Eventos
- Arquitecturas de Sistemas Distribuidos
  - Cliente-Servidor
  - Objetos Distribuidos
  - Peer-To-Peer
  - Service Oriented Architecture (SOA)

Shared-Data

- Generalidades
  - Hay un almacenamiento central de datos y un conjunto de componentes que operan sobre éste.
  - Interacción - intercambio de datos persistentes
  - Múltiples accesos a los datos
  - Al menos un almacenamiento compartido
  - Si se avisa al consumidor de nuevos datos de interés es un Pizarrón (Blackboard)
  - Si es el consumidor quien busca los datos es un Repositorio

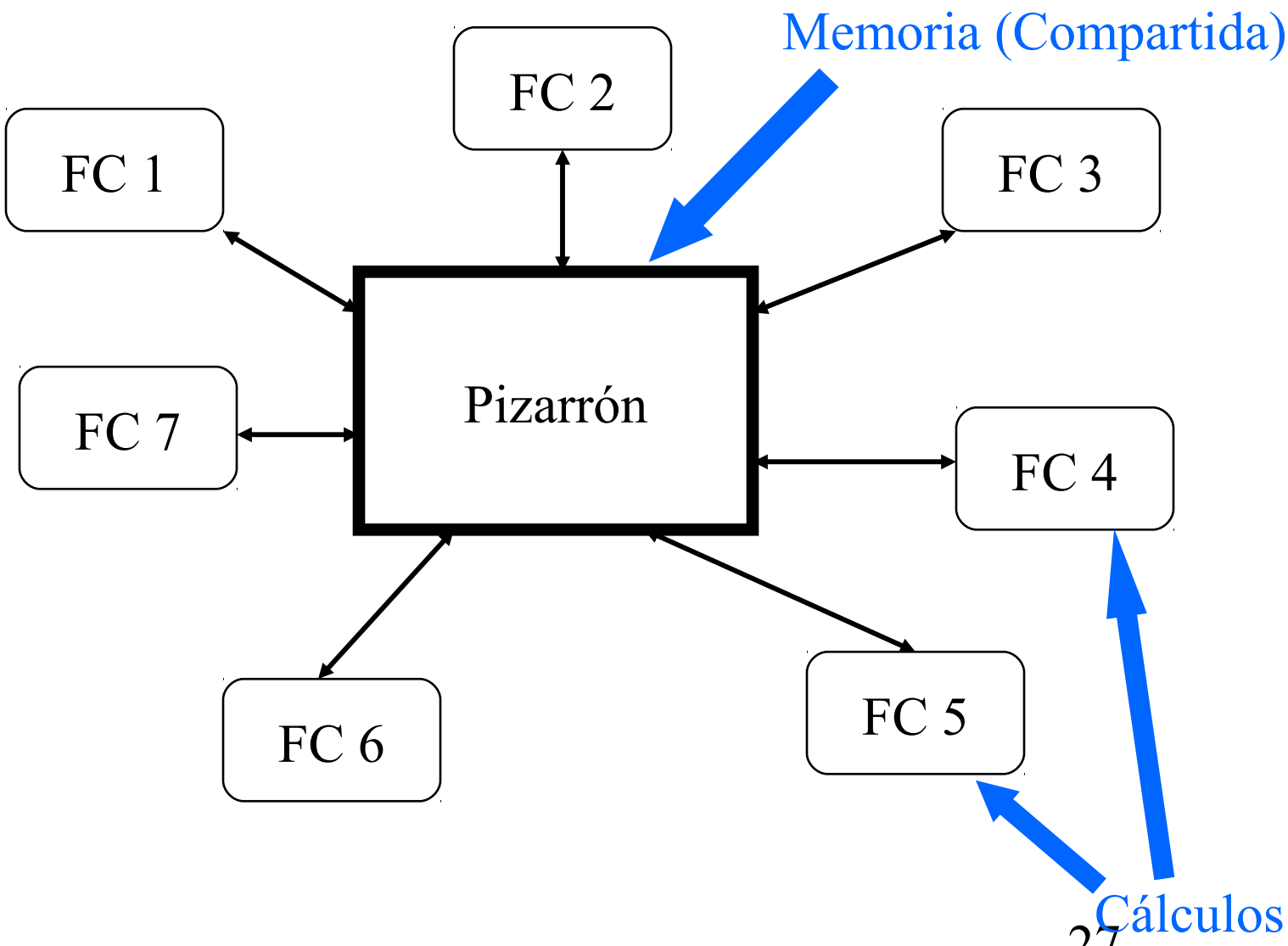
Ventajas y Desventajas

- Forma eficiente de compartir grandes cantidades de datos.
  - No se transmiten datos de un componente a otro.
- Sin embargo, los subsistemas deben estar de acuerdo en el modelo de datos del repositorio.
- Las componentes que producen datos no necesitan conocer como van a ser usados esos datos.
- Actividades como respaldos, seguridad, control de acceso y recuperación están centralizadas.
- Sin embargo, diferentes componentes podría tener distintas políticas de recuperación, respaldo, etc.

Pizarrón (Blackboard)

- Fuentes de Conocimiento
  - Procesos independientes que corresponden a particiones del conocimiento del mundo y del dominio dependientes de la aplicación
  - Responden a cambios en el pizarrón
- Estructura de datos del Pizarrón
  - Estado completo de la solución del problema
  - único medio por el cual las Fuentes de conocimiento interactúan para llegar a la solución
- Control
  - Guiado enteramente por el estado del pizarrón
  - Las Fuentes de conocimiento responden oportunamente cuando los cambios en el pizarrón aplican
  - Puede implementarse en las FC, en el pizarrón, en un componente separado o cualquier combinación de éstos.

Pizarrón (Blackboard)



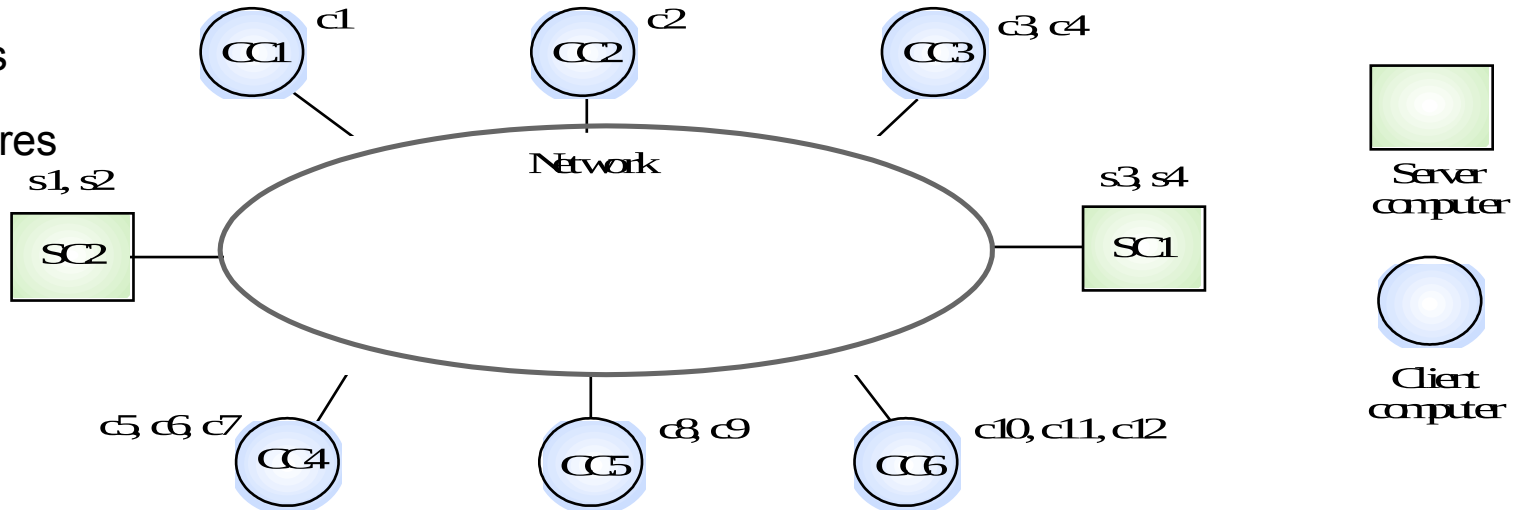
Cliente-Servidor

- El sistema se descompone en servicios y sus **servidores** asociados y en **clientes** que acceden y usan dichos servicios.
- Estilo compuesto por:
  - Servidores que ofrecen servicios.
  - Clientes que usan servicios ofrecidos por los servidores.
  - Una red que permite que los clientes accedan a los servidores.
    - Esto no es estrictamente necesario ya que clientes y servidores pueden estar en una misma máquina.
- Los clientes conocen a los servidores pero no a otros clientes y los servidores no tienen porqué conocer a los clientes
- la asignación de procesos a procesadores no tiene porqué ser 1:1

Cliente-Servidor (2)

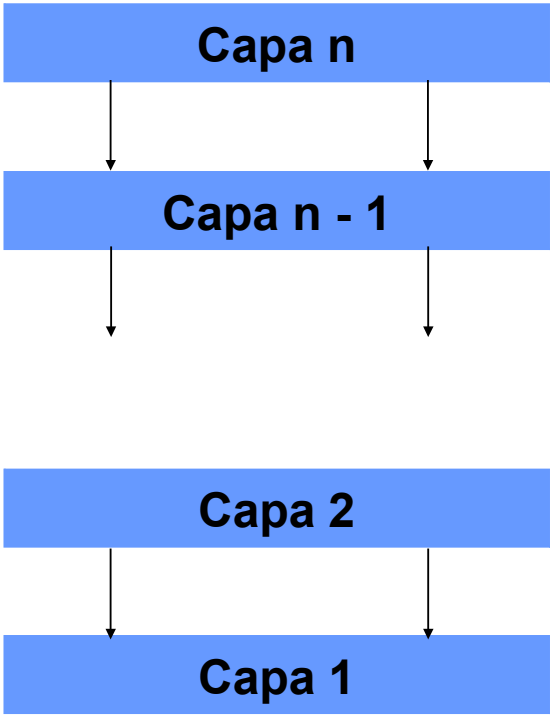
C_i = clientes

S_i = servidores



Capas Jerárquicas

- El sistema se organiza en capas.
- Cada una provee un conjunto de servicios a las capas superiores y requiere servicios de las inferiores.



Capas Jerárquicas (2)

- **Modelo estricto:** una capa sólo utiliza servicios de la inmediata inferior.
- **Modelo relajado**: se pueden “saltar” capas.
- Definición de protocolos mediante los que interactúan las capas

Ventajas y Desventajas

- Ventajas
  - Facilita la comprensión
  - Facilita mantenimiento
  - Facilita reutilización
  - Facilita portabilidad
- Desventajas
  - No siempre es fácil estructurar en capas ni identificar los niveles de abstracción a partir de los Requerimientos.
  - la interpretación de comandos en múltiples niveles puede afectar el desempeño.

Descomposición O.O.

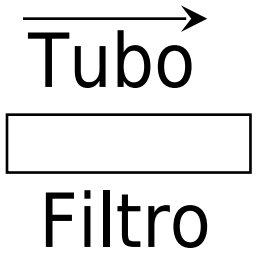
- Se descompone el sistema en un conjunto de objetos que se comunican.
- Representación de datos y operaciones asociadas se encapsulan en un objeto.
- Herencia, polimorfismo, sobrecarga de operadores, enlace dinámico.

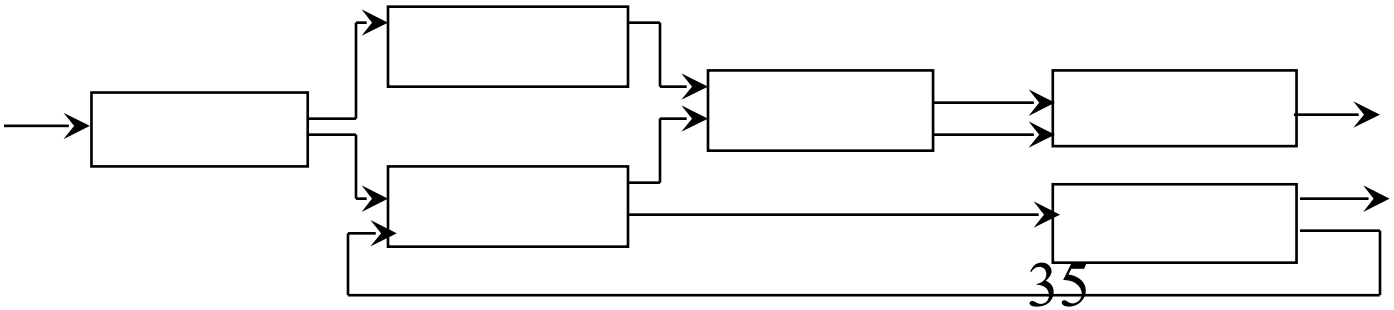
Ventajas y Desventajas

- Ventajas
  - La implementación de los objetos puede ser cambiada sin afectar a otros objetos.
  - Promueve la reutilización de componentes.
  - Muchos objetos representan entidades de la realidad por lo que es fácil entender la estructura del sistema.
- Desventajas
  - Para usar servicios se debe conocer el nombre de la interface de otro objeto.
  - Los cambios en las interfaces afectan a todos los objetos que la usan.

Tubos y Filtros

- Se descompone el sistema en módulos funcionales
- Interacción - sucesiva transformación de flujos de datos
- Los datos llegan a un filtro, se transforman y son pasados a través de tubos al siguiente filtro
- Un único filtro puede pasar datos a múltiples tubos y recibir datos de múltiples tubos
- Cada filtro es independiente del resto y no conoce la identidad de los otros filtros
- La transformación del filtro puede comenzar antes de terminar de leer la entrada
- Respetando el grafo, no importa la secuencia (paralelismo)





Ventajas y Desventajas

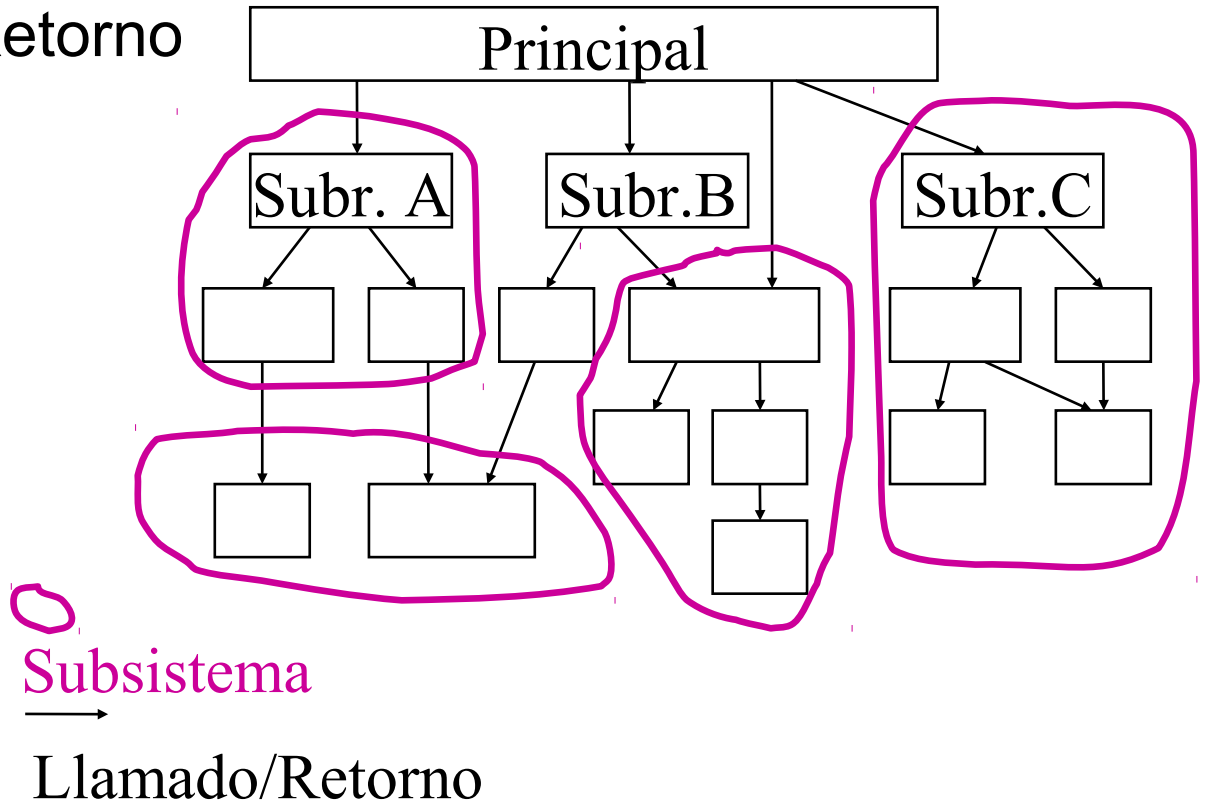
- Ventajas
  - Se pueden reutilizar los filtros.
  - Es intuitivo pensar en secuencias de procesamientos de datos.
  - Agregar nuevas transformaciones de forma que el sistema evolucione es sencillo.
- Desventajas
  - Acordar cuál es el formato de los datos.
    - Tiene que ser “genérico” si las componentes son reusadas.
  - Sistemas interactivos son difíciles de construir con este estilo.
  - Ineficiente si hace más de lo que debe o si los filtros repiten chequeos

Modelos de Control

- Modelos de control a nivel de la arquitectura se preocupan del flujo de control entre subsistemas
  - Control centralizado
    - Un subsistema controla al resto
  - Control basado en eventos
    - Cada subsistema puede responder a eventos externos
      - Eventos generados por otros subsistemas
      - Eventos generados por el ambiente del sistema

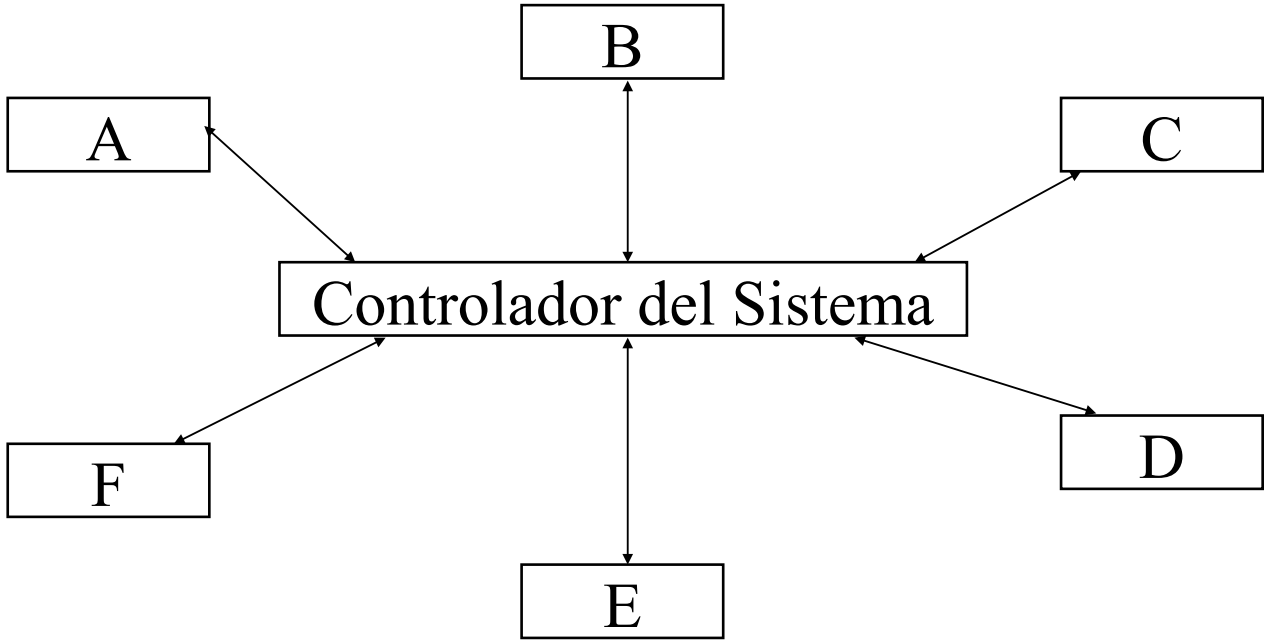
Control Centralizado (1)

- El control centralizado se puede dividir en dos clases
  - Llamada-Retorno



Control Centralizado (2)

- Modelo Gerente
  - Una componente es el gerente del sistema y controla a los otros procesos del sistema



Control Basado en Eventos

- En los modelos centralizados las decisiones de control suelen estar determinadas por valores de alguna variable de estado del sistema
- En cambio, el control basado en eventos es guiado por eventos generados externamente
- Ejemplos
  - Modelos emisión (broadcast). Se emiten los eventos a todos los subsistemas.
  - Modelos guiados por interrupciones. Se usan en sistemas de tiempo real. Las interrupciones son pasadas por un manejador de interrupciones a otras componentes para su procesamiento.

Publicar-Suscribir

- Sistema de componentes independientes que anuncian y se suscriben a eventos
- Interacción vía eventos anunciados
- Los componentes se suscriben a eventos
- Es trabajo del run-time asegurarse que cada evento publicado sea entregado a todos los suscriptores de dicho evento
- Una forma es la *invocación implícita*

Arquitecturas de Sistemas Distribuidos

El procesamiento de la información es distribuido entre varias computadoras.

- Ventajas
  - Se comparten recursos.
  - Apertura. Normalmente estos sistemas están diseñados con protocolos estándar.
  - Concurrencia.
  - Escalabilidad
  - Tolerancia a fallas

Arquitecturas de Sistemas Distribuidos (2)

- Desventajas
  - Complejidad
  - Seguridad
  - Difíciles de gestionar
  - Muy poco predecibles
- Ejemplos
  - Cliente-servidor
  - Objetos distribuidos
  - Peer-to-peer
  - Service oriented architecture (SOA)

Middleware

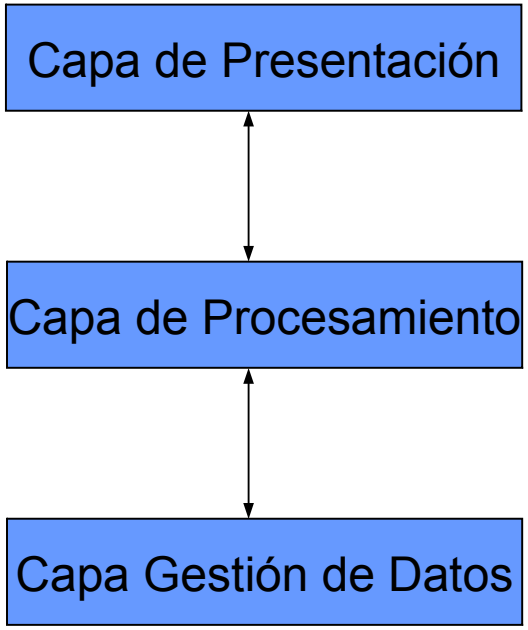
- Las componentes pueden ser
  - Implementadas en diferentes lenguajes
  - Ejecutarse en distintos tipos de procesadores
  - Usar diferentes modelos de datos
  - Representar de forma diferente la información
  - Usar distintos protocolos de comunicación
- Entonces, se necesita software para gestionar la comunicación y el intercambio de datos entre componentes
  - Dicho software se denomina *middleware*
    - Esta en el medio de las diferentes componentes distribuidas

Middleware (2)

- Normalmente son usados middleware que son comprados comercialmente ya que son de propósito general.

Cliente-Servidor

- El diseño de sistemas cliente-servidor debería reflejar la estructura lógica de la aplicación.
- Una forma de mirar a una aplicación es la siguiente:



Cliente-Servidor (2)

- Capa de presentación
  - Presentación de información al usuario e interacción con el mismo
- Capa de procesamiento
  - Implementación de la lógica de la aplicación
- Capa gestión de datos
  - Operaciones de bases de datos y archivos
- En sistemas distribuidos es importante distinguir claramente esta separación ya que es posible distribuir cada capa en computadoras diferentes

Cliente-Servidor en 2 Niveles

- Es la arquitectura más simple cliente-servidor
- Varios clientes y un servidor (o varios servidores idénticos)
- 2 formas diferentes:
  - Cliente fino
    - El procesamiento de la información y la gestión de los datos ocurre en el servidor. El cliente sólo es responsable de ejecutar el software de presentación.
  - Cliente grueso
    - El servidor es responsable sólo de la gestión de los datos. El cliente ejecuta el procesamiento de la aplicación (la lógica de la aplicación) y la presentación.

Cliente Fino, Grueso y Applet

- Cliente fino
  - Procesamiento pesado del lado del servidor
  - Procesamiento pesado en la red
- Cliente grueso
  - Mejor distribución del procesamiento
  - La administración del sistema es más compleja
    - Cuando el software de aplicación cambia hay que reinstalar la aplicación en cada computadora cliente
- Java Applets descargables
  - Están en el medio entre cliente fino y grueso
  - Luego de descargar el applet se ejecuta parte de la lógica de la aplicación en el cliente

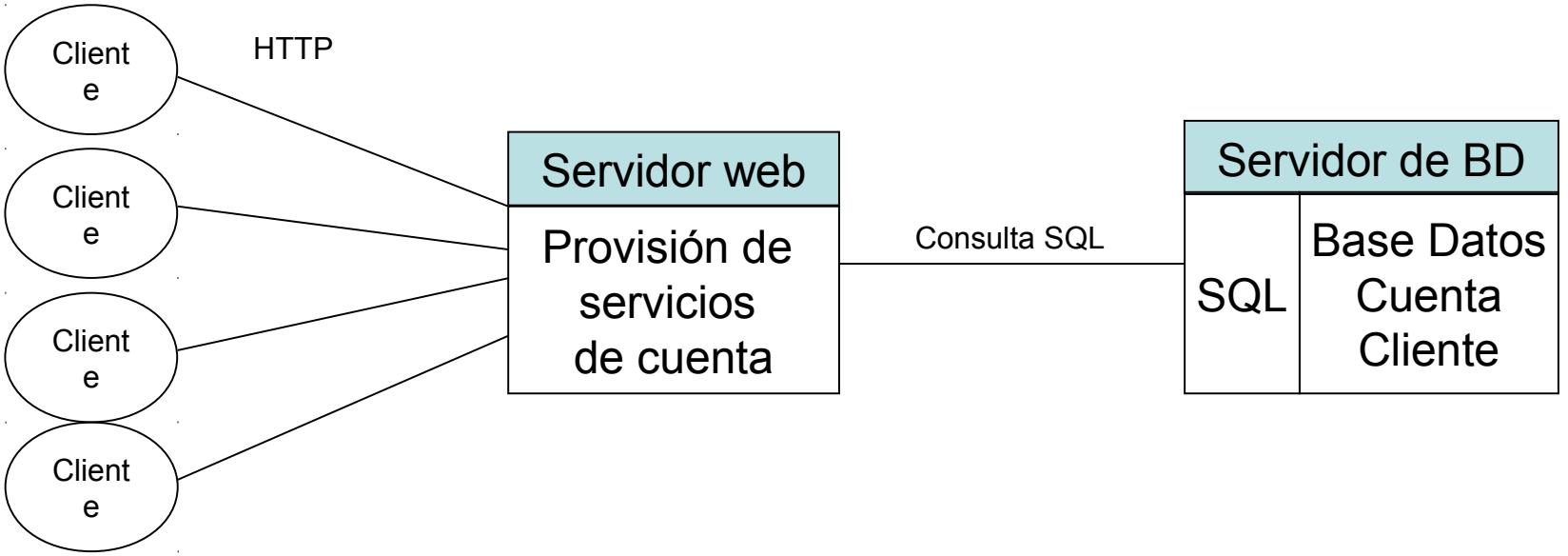
Algunos Problemas

- En 2 niveles hay que distribuir las tres capas (presentación, lógica y datos) en dos sistemas de computadoras
  - Pueden surgir problemas de
    - escalabilidad y rendimiento con el cliente fino
    - o de gestión del sistema si se usa cliente grueso

- Para evitar estos problemas una alternativa es usar una arquitectura cliente-servidor en 3 niveles

Cliente-Servidor en 3 Niveles

- La presentación, la lógica y los datos se separan como procesos lógicos diferentes distribuidos en distintas máquinas
- Ejemplo: Sistema bancario por internet



Comparación

- La arquitectura en 3 niveles
  - Es más escalable que la de 2 niveles
  - Reduce el tráfico en la red en comparación con cliente fino
  - La capa de procesamiento de la aplicación es fácilmente actualizada ya que está localizada centralmente
- Cliente servidor en múltiples niveles
  - Ejemplo: aplicación que necesita acceder a múltiples fuentes de datos (integración de datos)

Ejemplos

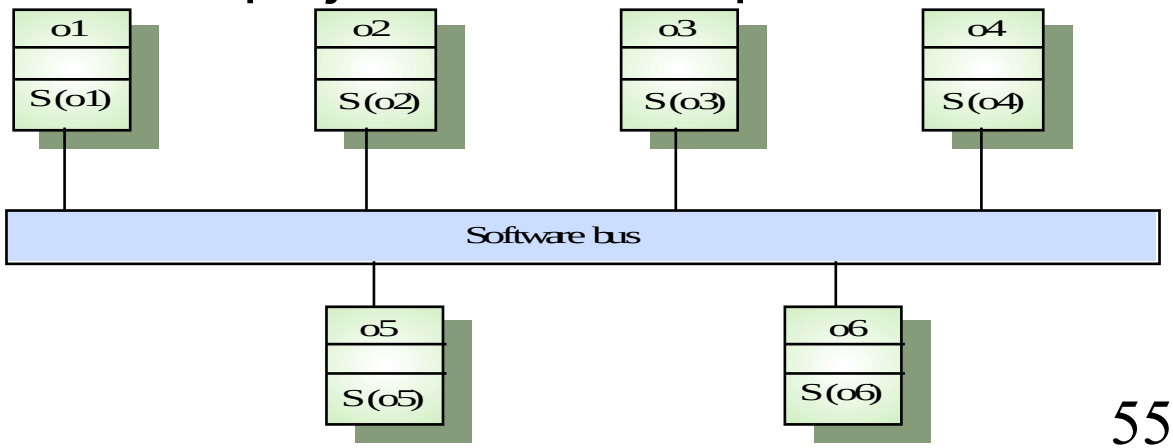
Arquitectura	Aplicaciones
Arquitectura C/S dos niveles cliente fino	• Aplicaciones legadas en las cuales no se puede separar el procesamiento de la gestión de los datos • Aplicaciones intensivas en las computaciones con poco o nada de gestión de datos (ej: compiladores) • Aplicaciones intensivas en manejo de datos (*browsing* y consultas) con poco o nada de procesamiento de aplicación
Arquitectura C/S dos niveles cliente grueso	• Aplicaciones donde el procesamiento de la aplicación es provisto off-the-shelf en el cliente • Aplicaciones donde se hacen intensivos procesamientos computacionales de los datos (ej: visualización de datos)
Arquitectura C/S tres niveles o múltiples niveles	• Aplicaciones de gran escala con cientos o miles de clientes • Aplicaciones en donde tanto los datos como el procesamiento son volátiles • Aplicaciones con fuentes de datos múltiples

Objetos Distribuidos

- En la arquitectura cliente-servidor los clientes y los servidores son distintos
  - Los clientes reciben servicios de los servidores y no de otros clientes
  - Los servidores pueden actuar como clientes de otros servidores pero no requieren servicios de clientes
  - Los clientes deben conocer los servicios ofrecidos por servidores específicos y deben conocer como contactar a estos servidores
- Enfoque más general
  - Remover la distinción entre clientes y servidores
  - Diseñar la arquitectura como una de objetos distribuidos

Objetos Distribuidos (2)

- Se compone de objetos que proveen servicios a otros objetos y usan servicios de otros objetos
  - No hay distinción entre clientes y servidores
  - Pueden ser distribuidos entre distintas computadoras en una red y se comunican a través de *middleware*
    - Este tipo de *middleware* se conoce como *Object Request Broker* (ORB), por ejemplo, CORBA.
  - Es más complejo de diseñar que cliente-servidor



Peer-to-Peer

- Sistemas descentralizados donde las computaciones pueden ser realizadas en cualquier nodo de la red
  - En principio no hay distinción entre clientes y servidores
  - El sistema se diseña para usar el poder de almacenamiento y procesamiento de múltiples computadoras en una red
  - Los protocolos que permiten la comunicación entre nodos están embebidos en la propia aplicación
    - Cada nodo debe correr una copia de la aplicación

Peer-to-Peer (2)

- Ejemplos
  - Sistemas para compartir archivos
  - Mensajería instantánea
  - Seti@home
    - Y otros @home
  - Se está comenzando a usar también en el área de negocios
    - Intel y Boeing han implementado sistemas intensivos en las computaciones con arquitecturas p2p
- Se pueden clasificar en sistemas descentralizados o semi-centralizados

Peer-to-Peer (3)

- Algunos problemas
  - Overhead
  - Seguridad
  - Confianza
- Entonces, son más usados en sistemas que no son críticos respecto a la información

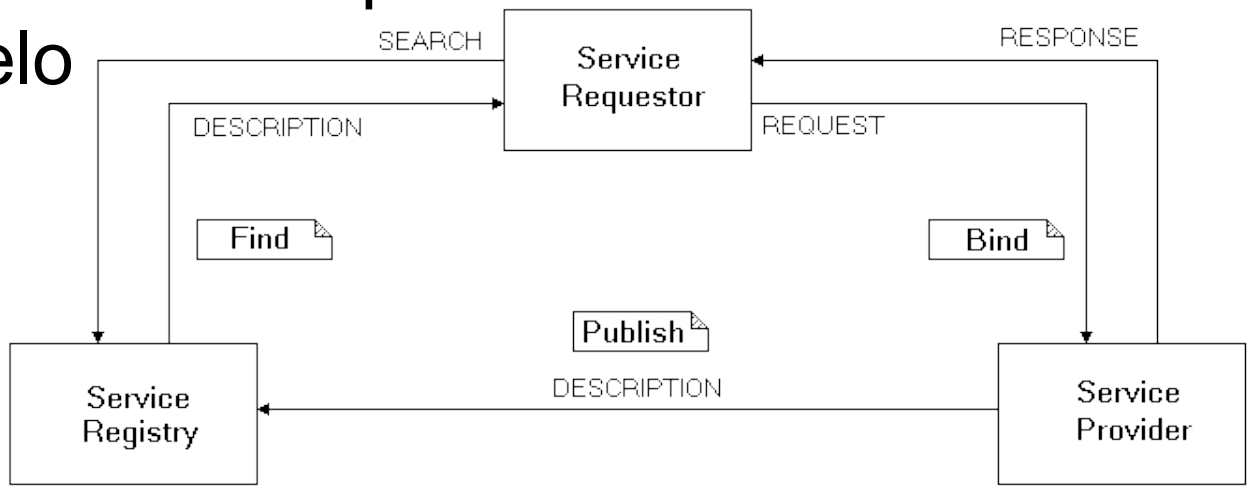
Service Oriented Architecture (SOA)

- Organizaciones que quieren hacer su información accesible a otros programas pueden hacerlo definiendo y publicando una interface de servicio web

- Un **servicio web** es una representación estándar para algún recurso computacional o de información que puede ser usado por otros sistemas

SOA (2)

- El servicio es independiente de las aplicaciones que lo usan
- Se pueden construir aplicaciones mediante el uso de distintos servicios
- Hay varios modelos de servicios distintos. Conceptualmente todos operan de acuerdo al siguiente modelo



Comparación SOA y Objetos Distribuidos

- Propaganda pública de la disponibilidad del servicio
- Potencialmente el enlace con el servicio puede ser en tiempo de ejecución
- Oportunidad de crear nuevos servicios a través de composición de otros servicios
- Pago por uso de servicios, por su uso más que por su provisión
  - Esto sustituye componentes caras raramente usadas
- Aplicaciones más chicas y compactas
- Aplicaciones que pueden ser reactivas y adaptar su operación de acuerdo a su medio mediante el cambio de enlaces a servicios durante la ejecución

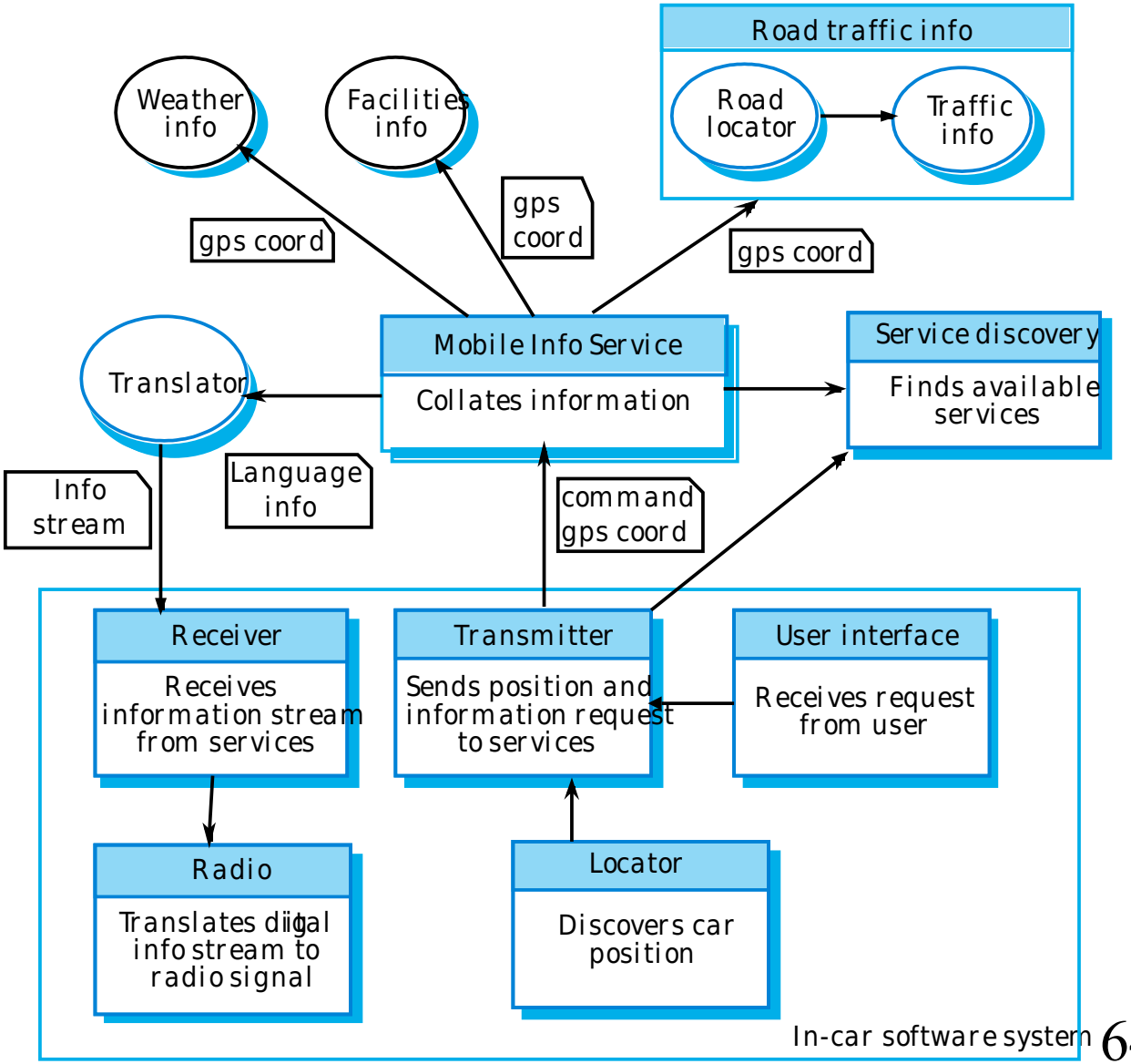
Web Services Estándares

- Los servicios se basan en estándares basados en XML
  - Entonces, pueden funcionar en cualquier plataforma y ser escritos en cualquier lenguaje
- Estándares más conocidos:
  - SOAP - Simple Object Access Protocol
  - WSDL - Web Services Description Language
  - UDDI - Universal Description, Discovery and Integration.

Ejemplo

- Un sistema de información para un auto provee al conductor con información acerca del clima, las condiciones del tráfico, información local, etc. Esto está vinculado con la radio del auto de forma que la información es entregada como una señal en un canal específico de la radio
- El auto está equipado con un recibidor GPS para conocer su posición y, basado en esa posición, el sistema accede a un rango de servicios de información. La información debe ser entregada en el lenguaje especificado por el conductor

Ejemplo (2)


Road traffic info
Road locator
Traffic info
Weather info
Facilities info
gps coord
gps coord
gps coord
Mobile Info Service
Collates information
Translator
Service discovery
Finds available services
Info stream
Language info
command gps coord
Receiver
Receives information stream from services
Transmitter
Sends position and information request to services
User interface
Receives request from user
Radio
Translates digital info stream to radio signal
Locator
Discovers car position
In-car software system

Evaluación de Arquitecturas

- Cambiar la arquitectura de un producto ya construido requiere mucho esfuerzo
- Entonces, es importante evaluar la arquitectura antes de implementarla completamente
- Verificar los requisitos de calidad establecidos
- Evaluaciones a posteriori resultan útiles como forma de aprendizaje y estudio de posibilidades de mejora, por ej. para una nueva versión del producto
- Software Engineering Institute (SEI) propone:
  - Architecture Tradeoff Analysis Method (ATAM)
  - Software Architecture Analysis Method (SAAM)