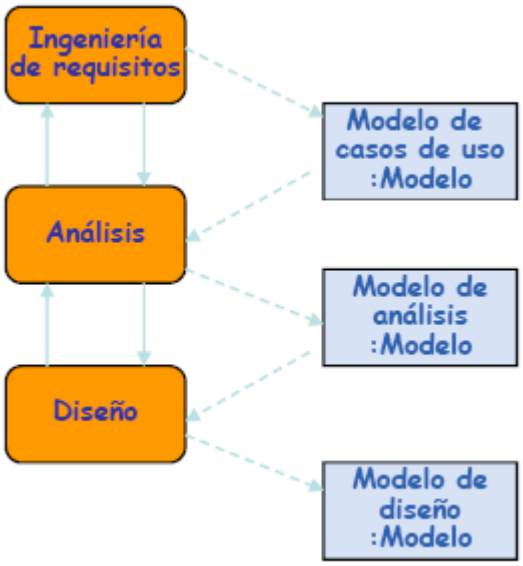
Tema 4. Diseño arquitectónico.

Introducción, Objetivos del Diseño.

Para la transformación del modelo de análisis en un **modelo de diseño** del sistema, se definen los objetivos de diseño del proyecto, se **descompone** el sistema **en subsistemas** más pequeños que pueden ser realizados por diferentes equipos y se **seleccionan estrategias para la construcción** del sistema como elegir la plataforma de hardware y software en la que se ejecutará, el formato y el sistema de almacenamiento de datos persistentes, la arquitectura estructural , el flujo de control global o la política de control de acceso e interfaz…



El modelo de diseño:

- Descripción clara de las estrategias.
- Descomposición en subsistemas.
- Diagramas que muestran la correspondencia entre hardware y software.
- Modelo de objetos que describe la realización física de los casos de uso.
- Muestra el impacto en el sistema de requisitos funcionales, no funcionales y restricciones.
- Sirve de abstracción de la implementación del sistema, convirtiéndose en la entrada fundamental de las actividades de implementación.

Ventajas del modelo de diseño:

- Reutilización a gran escala: posibilidad de tener partes ya hechas del sistema.
- Gestión de la complejidad: descomposición del problema.
- Herramienta de comunicación entre los participantes.
- Análisis más detallado del sistema.

Calidad y Diseño del software.

Un diseño de calidad proporciona representaciones del software en las que se puede **evaluar la calidad** del mismo, permite una "**traducción**" correcta **de los requisitos** en un programa y sirve como **fundamento para las actividades posteriores** (implementación, prueba y mantenimiento).

Sin diseño se corre el riesgo de construir un sistema inestable, no escalable y difícil de probar. Por norma general la falta de diseño provoca grandes dificultades en la gestión del proyecto y aumenta considerablemente el tiempo que se dedica a las pruebas.

El resultado de un proyecto sin diseño es la construcción de un sistema poco fiable que se escapa al control de sus creadores y que por lo tanto es difícil de corregir y mejorar, sistemas ineficientes que no optimizan los recursos y que posiblemente no se ajusten ni a las necesidades del cliente ni a las condiciones económico-temporales del proyecto.

Los sistemas sin un diseño de calidad suelen ser poco flexibles y por lo tanto difíciles de mantener, hasta un 70% del coste del proyecto se puede llegar a emplear en el mantenimiento del sistema.

Diseño Arquitectónico.

Los grandes sistemas siempre se **descomponen en subsistemas** que proporcionan conjuntos de servicios relacionados.

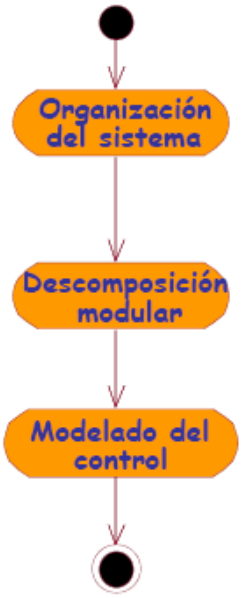
El proceso de diseño inicial que identifica estos subsistemas y establece como se lleva a cabo su control y comunicación se llama **diseño arquitectónico**.

Las actividades principales del Diseño arquitectónico son **decisiones**:

- **Estructuración del sistema** en varios subsistemas principales.
- **Descomposición modular** donde cada subsistema se divide en componentes o módulos interconectados.
- **Modelado del control** o estructuración de un plan de control para la ejecución del sistema por partes.

El diseño arquitectónico construye una salida que no es otra cosa que una serie de documentos con diversas perspectivas de la arquitectura del sistema:

- **Modelo estructural estático.** Describe subsistemas o componentes a desarrollar como unidades separadas.
- **Modelo de proceso dinámico.** Describe la organización del sistema en tiempo de ejecución.
- **Modelo de interfaz.** Describe la definición de los servicios ofrecidos por cada subsistema a través de su interfaz pública.
- **Modelos de relación.** Describe las relaciones entre los distintos módulos o subsistemas, por ejemplo: los flujos de datos entre subsistemas.
- **Modelo de distribución.** Describe como se distribuyen los subsistemas entre los componentes físicos (computadores, nodos de red…)



La arquitectura puede estar en función de **requisitos no funcionales** (rendimiento, robustez, mantenibilidad…) necesarios para el sistema y que en ocasiones pueden exigir arquitecturas contradictorias. Las principales condiciones no funcionales y sus "restricciones" son:

- **Rendimiento.** Si se necesita un elevado rendimiento se utilizarán pocos subsistemas con poca comunicación.
- **Protección.** Las aplicaciones con elevado nivel de seguridad necesitarán estructurarse en capas con los recursos críticos protegidos en las capas más internas y contarán con elevados niveles de validación.
- **Disponibilidad.** Puede obligar a incluir componentes redundantes que puedan reemplazarse y actualizarse sin detener el sistema.
- **Mantenibilidad.** Mejora cuando se utilizan componentes más pequeños que pueden intercambiarse con facilidad.

Organización del sistema. Arquitectura.

Definición genérica para todos los modelos: modelo arquitectónico para la estructuración del sistema.

La estructuración u organización se basa en la **identificación de subsistemas o capas** clave **a desarrollar de forma independiente** y en las **relaciones entre subsistemas**. Resulta efectivo para la comunicación entre los participantes en el proyecto y para realizar el **reparto de tareas** entre distintos grupos o recursos.

Los modelos organizacionales más usados son:

- **Modelo de depósito o repositorio.**
- **Modelo cliente-servidor.**
- **Modelo de capas o máquina abstracta.**

Organización: Modelo de repositorios.

Arquitectura en la que todos los datos compartidos se ubican en una base de datos central a la que acceden todos los subsistemas. Por ejemplo Ubuntu con su gestor de actualizaciones.

Resulta útil en sistemas que emplean grandes cantidades de datos, generados normalmente por un subsistema y empleados por otro.

Ventajas:

- **Compartición eficiente.** Se comparten grandes cantidades de datos sin necesidad de transmitir datos explícitamente de un subsistema a otro.
- **Ligera abstracción el manejo de datos.** Los subsistemas que producen datos no necesitan saber cómo son utilizados por otros subsistemas.
- **Centralización.** Centralización de actividades de administración del repositorio (respaldo, seguridad, control de acceso y recuperación de errores).
- **Integración directa.** Las herramientas compatibles con el modelo de datos se intregran directamente.

Desventajas:

- **Modelo de datos común.** Los subsistemas deben utilizar un mismo modelo de datos que el que esté implementado en el repositorio. Este “compromiso” entre las necesidades específicas de cada herramienta puede afectar a diversas cuestiones, entre ellas al rendimiento.
- **Difícil integración de subsistemas “externos”.** Resulta muy difícil o incluso imposible integrar subsistemas (normalmente heredados) cuyos modelos de datos no se ajusten al esquema del repositorio.
- **Dificulta la evolución.** Genera un gran volumen de información y es difícil hacer evolucionar el sistema.

- **Estandarización de las políticas.** Diferentes subsistemas pueden tener diferentes políticas de seguridad, de recuperación y respaldo pero el repositorio impone la misma política a todos los subsistemas.

- **Dificultad para distribuir.** Difícil distribuir el repositorio en varias máquinas por los posibles problemas de inconsistencia o de redundancia de datos.

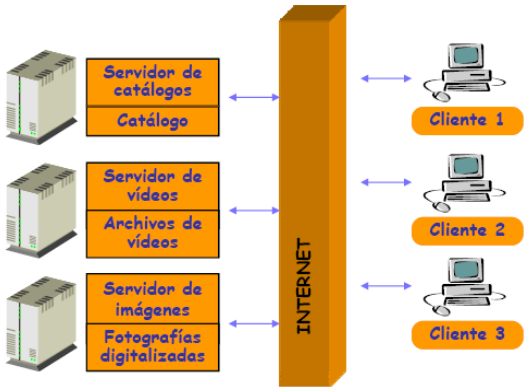
Organización: Modelo Cliente-Servidor.

Este modelo de sistema se organiza como un conjunto de servicios y servidores asociados junto con los clientes que acceden y usan dichos servicios. Por ejemplo un banco.

Componentes:

- **Conjunto de servidores.** Servidores independientes que ofrecen servicios a otros subsistemas (servidores de impresión, de administración de archivos…).

- **Conjunto de clientes.** Los clientes invocan los servicios ofrecidos por los servidores mediante un protocolo de petición-respuesta como http o www. Normalmente los clientes conocen el nombre de los servidores disponibles y los servicios que suministran pero los servidores no tienen porque conocer al cliente. Pueden existir varias instancias de un programa que se ejecutan de forma concurrente.



- **Una red.** Un sistema de comunicación que permita a los clientes acceder a los servicios (no es estrictamente necesario).

La ventaja más importante de este modelo es que es un **modelo de sistemas distribuido** que muestra como datos y procesamiento se pueden distribuir a lo largo de varios procesadores, es decir, **no existe una relación 1:1** entre procesos y procesadores. Un computador puede ejecutar varios procesos servidores o varios procesos clientes.

El diseño debe reflejar la estructura lógica de la aplicación. Dicha estructura suele ser de **tres capas** para una aplicación distribuida.

- **Capa de presentación.** Esta capa se encarga de mostrar la información e interactuar con el usuario.

- **Capa de procesamiento de la aplicación.** Esta capa implementa la lógica de la aplicación.

- **Capa de administración de datos.** En esta capa se realizan todas las operaciones de la base de datos.

Modelo cliente-servidor en Dos capas.

La arquitectura **cliente-servidor más simple** es un modelo en dos capas:

La aplicación se organiza como un servidor (o varios idénticos) y un conjunto de clientes.

- **Modelo de "cliente delgado."** Todo el procesamiento de la aplicación y la administración de datos se realiza en el servidor. El cliente únicamente es responsable de la presentación del os datos. Por ejemplo sistemas basados en tecnologías web, dispositivos sencillos de red…

  Ventajas:

  - Se suele utilizar cuando se heredan sistemas centralizados.
  - La interfaz migra a los PC's.
  - La aplicación misma actúa como servidor y maneja todo el procesamiento y la administración de los datos.

  Desventajas:

  - Implica una **gran carga de procesamiento** para el servidor.
  - El servidor realiza todos los cálculos, lo que provoca una **gran cantidad de tráfico** en la red entre el cliente y el servidor.
  - **Desaprovecha** la capacidad de cálculo de las **máquinas** de los procesos **cliente**.

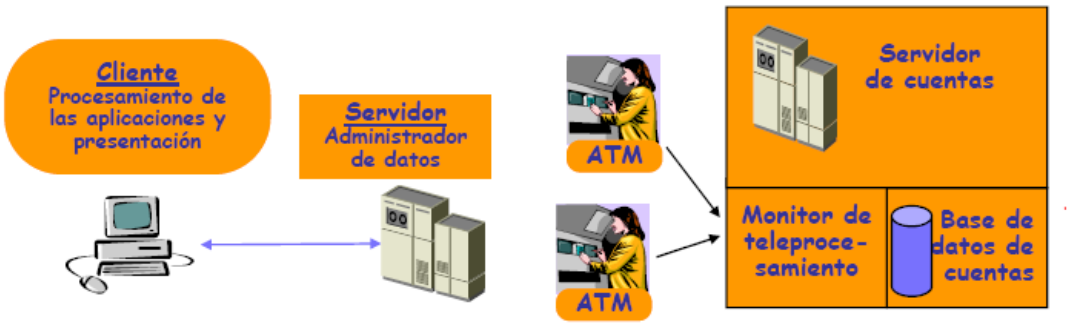
- **Modelo de "cliente grueso".** El servidor sólo es responsable de la administración de datos. El software del cliente implementa toda o gran parte de la lógica de la aplicación y las interacciones del usuario con el sistema.

  Ventajas:

- **Distribuye** al cliente **el procesamiento lógico** y la presentación.
- **Aprovecha la capacidad** de procesamiento **de los clientes**.
- **Los ATM no se conectan directamente a la base de datos** del cliente sino al gestor de transacciones.

Desventajas:

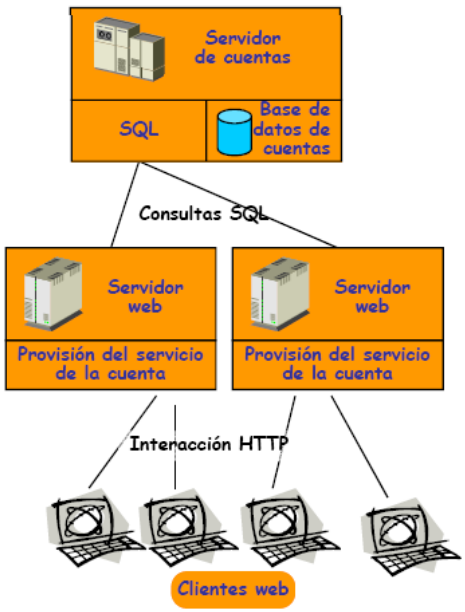
- **Administración** del sistema **más compleja** al distribuirse la funcionalidad de la aplicación.
- **Aumenta el coste del mantenimiento** ya que es necesario la reinstalación o actualización de cada computador si la aplicación cambia.



Los **problemas** de este enfoque en dos capas son que las **tres capas lógicos** (presentación, procesamiento y administración de datos) deben asociarse a **dos sistemas de cómputo**. El modelo de "cliente delgado" es poco escalable y su rendimiento es menor en contraposición a los problemas de administración del como de "cliente grueso".

Modelo cliente-servidor en Tres capas.

Las tres capas son procesos separados lógicamente que se ejecutan en procesadores separados. Son más escalables que las arquitecturas en dos niveles.



Componentes:

- **Administración de datos.** Suministrado por la base de datos (normalmente es un mainframe).
- **Servicios de aplicación.** Suministrados por un servidor Web.

- **Presentación.** El cliente es el computador del usuario con un sistema de presentación o interfaz.

La principal ventaja con respecto al sistema en dos capas es que es **escalable de forma sencilla** ya que se pueden añadir sin problemas más servidores web que aumenten la capacidad de soporte de clientes simultáneos en el sistema.

Arquitectura	Aplicaciones
C/S de dos capas con clientes delgados	Aplicaciones de sistemas heredados donde no es práctico separar el procesamiento de las aplicaciones y la administración de datos Aplicaciones computacionalmente intensivas como los compiladores con poca o ninguna administración de datos Aplicaciones intensivas en datos (navegar y consultar) con poco o ningún procesamiento de la aplicación
C/S de dos capas con clientes gruesos	Aplicaciones con procesamiento de datos computacionalmente intensivo (por ejemplo, visualización de datos, animaciones gráficas,...) Aplicaciones con funcionalidad para el usuario final relativamente estable utilizadas en un entorno con administración de sistemas bien establecido
C/S de tres capas o múltiples capas	Aplicaciones de gran escala con cientos o miles de clientes Aplicaciones donde tanto los datos como la aplicación son volátiles Aplicaciones donde se integran datos de diversas fuentes

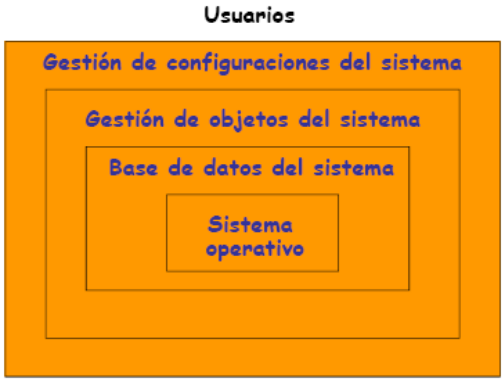
Organización: Modelo de capas (máquina abstracta).

Modela la interacción entre subsistemas mediante una organización en capas, **cada capa presta servicios** a la capa inmediatamente superior y actúa como cliente de la inferior (en la que queda encerrada). Por ejemplo las arquitecturas de red OSI y TCP/IP.

El diseño incluye los **protocolos** que establecen cómo interactuará cada par de capas.

Una de las mayores ventajas del modelo por capas es que es su arquitectura es cambiable y portable:

- Preservando la interfaz, una capa se puede reemplazar por otra.
- Cuando cambian las interfaces de las capas sólo afecta a las capas adyacentes.
- Sólo hay que reimplementar las capas más internas.



Modelo de capas de un sistema de gestión de versiones

Desventajas:

- Resulta difícil estructurar los sistemas pues es posible que el usuario requiera acceso a capas internas lo que subvierte el modelo.
- El rendimiento puede resultar afectado por los múltiples niveles de interpretación de órdenes que se requieran.

Descomposición modular.

Después de diseñar la arquitectura estructural se descomponen los subsistemas en módulos. No existe una distinción rígida entre la descomposición del sistema y la descomposición modular, se pueden aplicar los modelos arquitectónicos de forma recursiva; sin embargo, los componentes en los módulos son más pequeños que los subsistemas por lo que se utilizan modelos alternativos de descomposición.

Modelos principales de descomposición modular.

Modelo orientado a objetos.

- El sistema se descompone en un conjunto de objetos que se comunican entre ellos.
- Los módulos son objeto con estado privado y operaciones definidas sobre ese estado.
- Entidades reales fácilmente comprensibles, mejor **reutilización**.
- Difícil representar entidades más complejas.

Modelo de flujo de datos (o estructurado).

- El sistema se descompone en módulos funcionales que reciben datos y los transforman en datos de salida.
- Los datos fluyen de una función a otra y se van transformando.
- Incluye información sobre la **secuencia de operaciones** y resulta intuitivo para ciertas personas.

Modelado de Control.

Representa la forma en que los subsistemas se controlan para que sus servicios se entreguen en el lugar correcto y en el momento justo.

El **arquitecto** organiza los subsistemas de forma acorde a un modelo de control.

Existen dos modelos de control genéricos:

- **Control centralizado.** Un subsistema es el responsable de controlar, iniciar y detener otros subsistemas. También puede pasar el control a otros subsistemas pero espera que se le devuelva esa responsabilidad.

- **Control basado en eventos.** Cada subsistema puede responder a eventos generados en el exterior provenientes de otros subsistemas o del entorno del sistema.

Complementan los modelos estructurales siendo aplicable tanto un control centralizado como uno orientado a eventos.

Modelado de Control: Control centralizado.

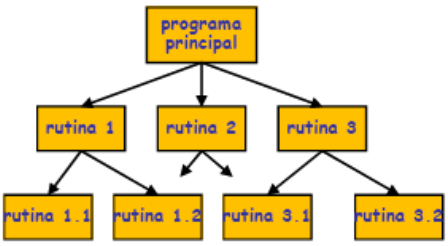
Un subsistema tiene la responsabilidad de controlar el sistema y administrar la ejecución de otros subsistemas. Existen dos clases, en base al modo en que se ejecutan los subsistemas, secuencial o paralelo.

Modelo de llamada-retorno (ejecución secuencial).

- El control se inicia en la parte superior de una jerarquía y por medio de llamadas a subrutinas pasa a los diferentes niveles del árbol.

- No es un modelo estructural por lo que no es necesario que, por ejemplo, la Rutina 1.1 forme parte de la Rutina 1.

- Utilizado por lenguajes de programación como Ada, Pascal y C aunque también con lenguajes Orientados a Objetos (OO).



Ventaja:

- Es relativamente **sencillo analizar los flujos de control** y conocer cómo responderá el sistema a cierto tipo de entradas.

Inconveniente:

- Las **excepciones** a operaciones normales son **complicadas de gestionar**.

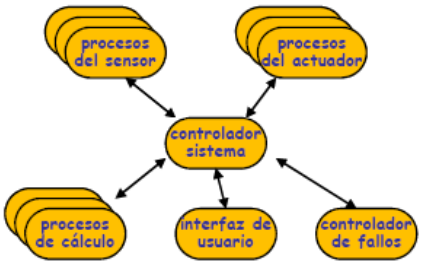
Modelo del administrador (concurrente).

- Un componente del sistema se designa como administrador y controla el inicio, detención y coordinación del sistema según las variables de estado del sistema, verifica si otros procesos han producido información para procesar o si ha de pasarles información para el procesamiento.

- Un proceso es un subsistema o módulo que se ejecuta en paralelo con otros procesos.

- Utilizado en sistemas de tiempo real "suaves", es decir, con restricciones de tiempo no muy estrictas.

- Llamado también **modelo de ciclos de eventos**.



Modelado de Control: Control dirigido por eventos.

Se rigen por eventos generados en el exterior como pueden ser: la señal de un sensor, un comando desde un menú… Un ejemplo de este tipo de sistemas son las hojas de cálculo.

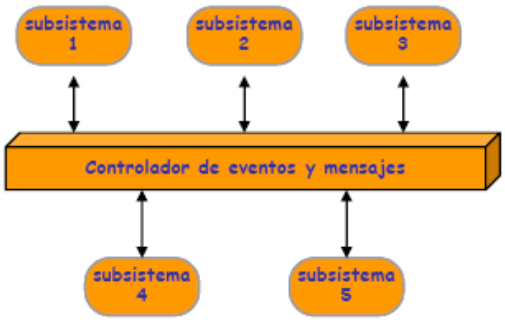
Existen dos tipos de modelos dirigidos por eventos (principales tipos):

- **Modelos de transmisión (broadcast).** Los subsistemas registran un interés en eventos específicos (se suscriben) y cuando ocurren esos eventos el control se transfiere al subsistema que puede manejar el evento.

- **Modelos dirigidos por interrupciones.** Especialmente útiles para sistemas de tiempo real que necesitan manejar rápidamente eventos generados desde el exterior.

Modelado de Control: Control por eventos: Modelos de Transmisión.

Se diferencia del control centralizado en que la política de control no está contenida en el controlador de eventos y mensajes, sino que los s**ubsistemas deciden qué eventos requieren y el controlador asegura que estos eventos sean enviados** a dichos subsistemas.



Resultan efectivos para integrar subsistemas distribuidos a lo largo de diferentes computadores de una red. También son utilizados por los **agentes de solicitud de objetos (ORBs)** para comunicaciones de objetos distribuidos.

Ventajas:

- La evolución es relativamente sencilla pues se pueden integrar nuevos subsistemas registrando sus eventos en el controlador de eventos.

- Cualquier subsistema puede activar otros subsistemas sin conocer su nombre o ubicación.

- Los subsistemas se pueden implementar en máquinas distribuidas de forma transparente para otros subsistemas.

Desventajas:

- Los subsistemas no saben si los eventos se manejarán ni cuando lo harán.
- Cuando un subsistema genera un evento no sabe que otros subsistemas han registrado un interés en ese evento.

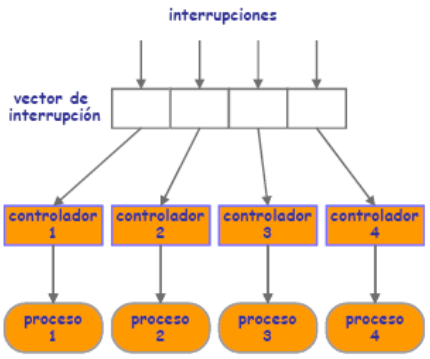
Modelado de Control: Control por eventos: Modelos dirigidos por Interrupciones.

Resultan útiles para **sistemas de tiempo real** que necesitan manejar muy rápidamente eventos generados en el exterior como por ejemplo sistemas de seguridad en automóviles.

Se pueden combinar con el modelo de administrador centralizado de manera que el administrador central maneja la ejecución normal del sistema utilizando el control basado en interrupciones para casos de emergencia.

Interrupciones.

- Existen varios tipos de interrupciones conocidas con un controlador definido para cada tipo.
- Cada tipo de interrupción se asocia con la ubicación de memoria en la que se almacena la dirección del controlador.
- Cuando se recibe una interrupción de un determinado tipo, un interruptor de hardware transfiere el control al controlador adecuado.
- El controlador puede iniciar o detener otros procesos en respuesta a los eventos recibidos por el interruptor.



Ventajas:

- Permite dar respuestas muy rápidas a los eventos.

Desventajas:

- Complejo de programar y difícil de validar (replicación de ocurrencias)
- Si el número de interrupciones está limitado por el hardware, cuando se alcanza el límite no se pueden gestionar más tipos de eventos. Se pueden asignar distintos

tipos de eventos a una interrupción, dejando que el controlador detecte que evento ha ocurrido pero disminuye el rendimiento.

Sistemas Distribuidos.

Todos los grandes sistemas informáticos son en la actualidad sistemas en los que **el procesamiento** de la información **se distribuye sobre varias computadoras**.

Proporciona **ventajas** como: compartición de recursos, apertura, concurrencia, escalabilidad y tolerancia a defectos.

Las principales **desventajas** se centran en la complejidad, la seguridad, manejabilidad e impredecibilidad del sistema.

Por seguridad e interoperabilidad se ha utilizado sobre todo computación distribuida intraorganizacional, es decir, servidores dentro de una misma organización donde resulte sencillo aplicar estándares locales y procesos operacionales.

Un ejemplo de sistema distribuido es el p2p.

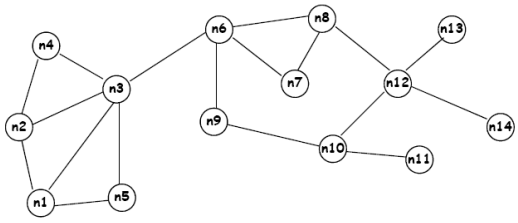
Sistemas peer-to-peer (p2p).

Son sistemas descentralizados donde los cálculos se pueden realizar en cualquier nodo de la red, no se distingue, a priori, entre clientes y servidores.

Estos sistemas están diseñados para aprovechar la ventaja de potencia computacional y disponibilidad de almacenamiento de grandes redes.

Utilizan estándares y protocolos de comunicaciones embebidos en la propia aplicación y cada nodo ejecuta una copia de la misma.

Ejemplos de p2p como: Kazza, eMule, Messenger, ICQ…



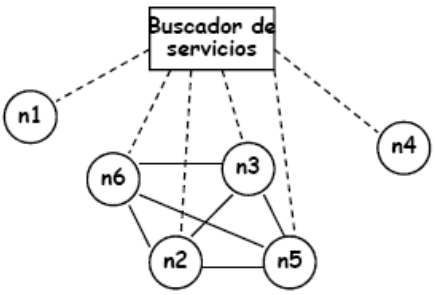
Comienza a utilizarse en entornos corporativos aunque uno de sus principales problemas es la falta de protección, autentificación…

Teóricamente cada nodo en la red puede conocer a cualquier otro nodo pero, como es inviable, se organizan por “localidades” con nodos-puente entre ellas.

Existen dos arquitecturas p2p:

- **Arquitectura descentralizada.** Muy redundante y por lo tanto **tolerante a fallos** y a desconexiones de nodos aunque puede sufrir **sobrecargas** y replicaciones de comunicación.

- **Arquitectura semicentralizada.** Los nodos actúan como servidores para facilitar las comunicaciones, una vez



localizados los nodos se establecen conexiones directas y no es necesario el servidor.

Sistemas de Sistemas Orientados a Servicios.

El desarrollo de la www permitió que se pudiera acceder a información de otras organizaciones bajo el formato HTML.

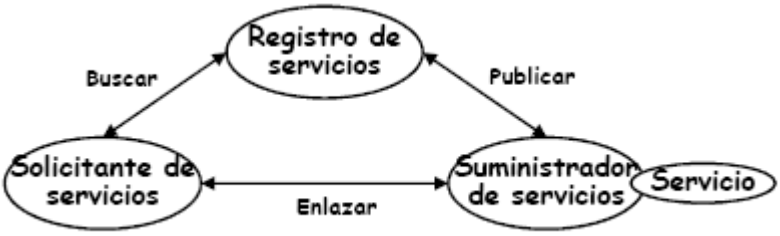
Servicio web.

- **Representación estándar** para cualquier recurso computacional o de información que pueda ser usado por otros programas.
- Permite que la provisión de un **servicio** sea **independiente** de la aplicación que lo utiliza.
- Las organizaciones pueden **hacer accesible información** a diferentes programas definiendo y publicando una **interfaz de servicio web** que defina los datos y su forma de acceso.

Componentes de un servicio web:

- **Proveedor de servicios.** Desarrollan y ofertan servicios a usuarios y permiten construir aplicaciones enlazando servicios de diferentes proveedores.
- **Solicitante del servicio.** Enlaza este servicio a su aplicación, incluye código para llamar al servicio y procesa el resultado.

Diferentes tipos que se ajustan al mismo modelo:



Ventajas:

- Los usuarios pueden pagar por los servicios sólo en función del uso. No hay por qué comprar componentes caros que rara vez se utilicen.
- Aplicaciones más pequeñas (manejos de excepciones como servicios externos).
- **Construcción a medida** de nuevos servicios, enlazando servicios existentes.

Estándares basados en XML:

- **SOAP** (Simple Object Access Protocol). Define una organización para intercambio de datos estructurados entre servicios web.
- **WSDL** (Web Services Description Language). Define como pueden representarse las interfaces de servicios web.
- **UDDI** (Universal Description, Discovery and Integration). Estándar de búsqueda que define como puede organizarse la información de descripción de servicios.