Diseño de Sistemas Distribuidos

Enric Martínez Gomáriz

Parte 1 - Introducción a los Sistemas Distribuidos

Presentación

Este libro está dedicado **al Diseño de Aplicaciones en Sistemas Distribuidos** con los dos entornos posibles, **Sistemas Operativos convencionales e Internet**. En los sistemas informáticos ambas soluciones coexisten, se complementan y refuerzan mutuamente.

La primera parte está dedicada a:

- Presentar los componentes de un sistema distribuido.
- A que el lector que no conoce que es un Sistema y una Arquitectura Distribuidos y como impacta en los Sistemas de Información (SI), obtenga esa formación. Se fijan los conceptos y la terminología sobre los que se apoya el resto del libro.
- Introducir el modelo distribuido basado en la obtención de servicios en arquitectura cliente/servidor.
- Se presentan las dos implementaciones posibles de Cliente/Servidor en que se fundamenta la arquitectura: Sistemas Operativos e Internet, y se introducen los conceptos de ambos entornos que afectan al diseño de aplicaciones distribuidas.
- Se introduce el concepto de servicio.

Además, es notoria la gran dispersión de terminología que se esconde detrás de los términos Cliente/Servidor e Internet. Ello hace necesario fijar una terminología clara para el desarrollo del método de diseño que se plantea a lo largo de la segunda parte del libro. Presentar esa terminología es también un objetivo de la primera parte

Otro objetivo fundamental de esa primera parte es definir una capa lógica que, sobre la capa física que proporciona la plataforma distribuida, permita diseñar aplicaciones trasparentes a las condiciones específicas de esa plataforma. Surgirá el concepto de **servicio** como pieza fundamental del diseño y a la arquitectura SOA (Arquitectura orientada a Servicios) como paradigma de diseño

Finalmente, se presentan y justifican los conceptos básicos del diseño y la administración.

La segunda parte está dedicada específicamente al Diseño. Se utilizan los conceptos y nomenclatura desarrollados y presentados en la primera parte.

Si Vd. ya conoce los fundamentos de una arquitectura distribuida sobre Sistemas Operativos e Internet, lea aquello que le parezca novedoso o de interés y sáltese lo demás. Pero por favor, en este caso intente coordinar su terminología con la mía. Le agradeceré ese esfuerzo, fundamental para en viaje por el diseño distribuido que iniciamos juntos.

La tercera parte desarrolla un ejemplo completo con ampliaciones que se proponen como trabajo adicional para el lector.

Sistemas Distribuidos

1. ¿Nos situamos?

La generalización del termino **cloud computing,** la popular nube, como paradigma de todo tipo, tanto organizativo como de diseño de sistemas, comporta una interesante reflexión.

Si la nube permite a los clientes y usuarios poder obtener funcionalidades a través de servicios de los cuales solo conocen su contrato de servicio pero ignoran el diseño y la localización, ¿para qué leer un documento como el que tiene entre las manos?

Figura 1. La doble visión

Muchas veces olvidamos que los servicios han de ser fabricados, y que para eses trabajo, hay que prepararse y hacerlo bien, muy bien, ya que nuestros clientes son en la mayoría de los casos desconocidos y si nuestro producto no es correcto, simplemente nos dejaran.

Así pues, por encima de la nube están los usuarios i clientes, tanto finales como los profesionales que reutilizan los servicios, y por debajo, los constructores y suministradores de esos servicios.

Esta doble visión, no excluyente ya que los constructores pueden ser a si mismo clientes cuando reutilizan servicios, estará presente en todo el documento que tiene entre manos.

2. Bienvenidos a los Sistemas Distribuidos.

Vamos a iniciar un viaje con un objetivo final: el diseño de aplicaciones distribuidas. Pero, cuando hablamos que diseñar un sistema distribuido, ¿de qué estamos hablando?

> Un sistema distribuido es un sistema de información en el cual las funciones se reparten por áreas de trabajo diferentes que trabajan de forma coordinada para asumir los objetivos que la organización asigna a ese sistema de información.
>
> Esta definición no obliga a que los servicios sean internos ni fabricados por la propia organización.

En él se integran.

- Los **objetivos de la empresa**. No olvidemos que son la justificación de la existencia de la Informàtica.
- La **plataforma de proceso**. Una vez diseñado el sistema, es el elemento

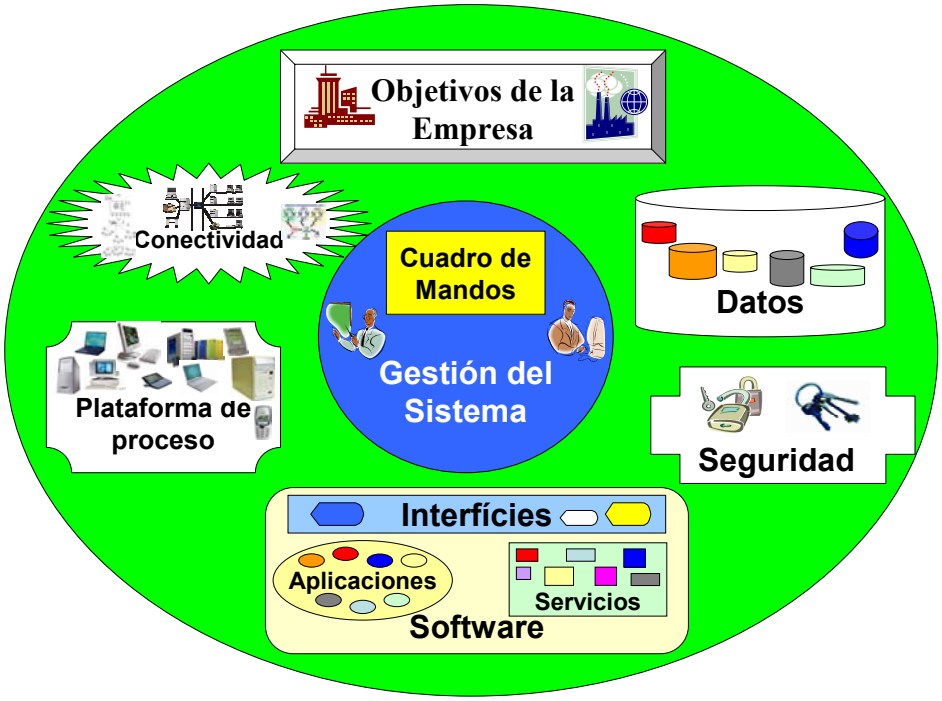


Figura 2. Elementos de un sistema Distribuido

encargado de proporcionar los recursos físicos y el software de base para ejecutarlo. Esta formado por los Mainframe, PC's, PDA's, teléfonos, etc...

- Los elementos de la **conectividad**. Son los encargados se proporcionar el transporte para comunicar e integrar los elementos de la plataforma de proceso. Son básicamente las redes y las comunicaciones.
- El **almacenamiento de datos**, formado por los datos en si y los gestores donde se localizan.
- Los elementos de **software** donde se incluyen las **aplicaciones,** los **servicios** que ayudan a crearlas y las **interfícies** que ayudan a usarlas. En este componente se integran las arquitecturas posibles para crearlas: centralizada, Batch, transaccional, cliente / servidor basado en sistema operativo, cliente / servidor basada en Internet y aplicaciones Web Internet. A lo largo de la exposición pondremos especial cuidado en presentar las características y posibilidades las tres últimas.
- **Sistemas de seguridad**.
- Finalmente, debe realizarse la **gestión del sistema** como un conjunto integrado y coordinado a través de los recursos de dirección y administración. La gestión del sistema debe permitir la coexistencia de varios centros de gestión diferentes. Parte fundamental del sistema de gestión es el **cuadro de mandos.** Hay dos cuadros de mandos diferentes:
  - El **cuadro de mandos de seguimiento de los objetivos de negocio** pensado para proporcionar información automática a los gestores de como la realidad se mueve respecto a las previsiones de los objetivos de negocio en "tiempo real".
  - El **cuadro de mandos de explotación** desde donde se centraliza y coordina toda la administración, supervisión y explotación del sistema.

Y todos ellos repartidos por varias plataformas físicas, distribuidas por compañías propias, clientes, proveedores y terceros con dispersión geográfica y desconocimiento mutuo de las plataformas respectivas.

Estos recursos técnicos suelen catalogarse en:

- Infraestructura.
  - Plataforma.
  - Comunicaciones.
- Datos.
- Software:
  - Aplicaciones.
  - Interfícies.
  - Servicios.
- Seguridad.

Pero no olvidemos que detrás del sistema operativo hay **personas** que lo usan y los gestionan. **El factor humano será fundamental** como nos cuidaremos de recordar a lo largo del todo el diseño.

Diseñar un sistema distribuido es crear aplicaciones de software que, utilizando **servicios** y ayudándose de la conectividad, participen y se integren en este entorno de forma transparente a las plataformas de proceso y de almacenamiento de datos, dotándolas de los recursos necesarios para gestionarse de forma integrada con el resto del sistema distribuido.

Los servicios permitirán usar todos los recursos técnicos y el sistema distribuido resultante no será nada más, ni nada menos, que un conjunto de servicios que interoperan entre ellos colaborando para cumplir los objetivos que se han establecido para el sistema.

Los sistemas distribuidos que se diseñen y construyan deben estar alineados con los objetivos de negocio de la empresa, aumentar la eficacia y eficiencia operacional de la compañía y permitir el mayor rendimiento con el menor coste en las estructuras informáticas que dan soporte.

No olvide nunca estos tres puntos. El **objetivo es siempre alinear tecnología y negocio**.

El sistema resultante debe ser adaptable, ofrecer el rendimiento necesario con el coste más barato que seamos capaces de conseguir.

Con este objetivo final, empezamos nuestro viaje para el cual le voy a pedir un esfuerzo. Las tecnologías llegan, se consolidan o desaparecen, y al final mueren. Y siempre con facilidad y rapidez.

Pero las estrategias, las tácticas y las técnicas de diseño tienen un ciclo de vida mucho más lento y robusto. Y están por encima de las tecnologías en que se implementan. Intente poner en su mochila solo las primeras. Este es un viaje por el mundo del diseño de sistemas distribuidos, no sus técnicas de implementación aunque haremos las necesarias salidas a ese mundo cunado sea necesario.

Espero de todo corazón que disfrute de este viaje y que cuando lleguemos al final piense que ha valido la pena.

3. Arquitecturas en un sistema distribuido. La arquitectura de Empresa.

La palabra arquitectura es de aquellos términos utilizados ampliamente dentro del mundo informático. Cuando atacamos sistemas distribuidos, la palabra aparece continuamente.

Esta constatación de la realidad no resulta extraña si acudimos a la definición que ANSI/IEEE hace del término: "**Arquitectura es la organización fundamental de un sistema, donde se integran sus componentes, se establecen las relaciones e interdependencias entre esos componentes y su entorno y se establecen los principios para su diseño, gestión y evolución**".

Así, en el mundo de los sistemas distribuidos donde conviven tantos factores y tan diferentes, es lógico que el término se utilice profusamente en varios lugares. Veamos la primera aparición.

El objetivo fundamental de cualquier sistema distribuido será aportar valor añadido. Y para empezar eficientemente el camino conviene organizar de alguna forma todos los factores que intervienen. La forma de hacerlo es proponer una **Arquitectura de Empresa**, conocida también por EA desde su nombre en inglés, *Enterprise Architecture*.

> La arquitectura de la empresa permite a la compañía conocer como es su estructura y la forma en que trabaja. Es el plano de ruta para el desarrollo de los negocios y de la tecnología que va ha apoyarlos, tanto en lo nuevo como en la evolución.

Es en este último aspecto por el que nos conviene acercarnos a ella. Sus contenidos son prerrequisitos que los sistemas distribuidos deberán cumplir. Es de aquí donde se origina el **Plan Estratégico Distribuido de la Compañía**, donde se registrarán todos los prerrequisitos de desarrollo y gestión que los sistemas distribuidos de la compañía deberán seguir y cumplir. Veremos que este documento se utilizará en la segunda parte dedicada al diseño, como base de muchas decisiones a tomar durante el desarrollo del sistema distribuido.

La arquitectura de empresa se articula sobre cinco enfoques.

3. 1. La Perspectiva de Negocios (Business Perspective).

Describe como trabaja la compañía. Incluye las relaciones con terceros y los planes de evolución desde el estado actual al objetivo deseado.

Son componentes clásicos de la perspectiva de negocio:

- Los procesos de negocio.
- Los **Manuales de Procedimientos**.
- Objetivos a corto, medio y largo plazo.
- Las estructuras organizativas y sus condicionamientos.
- Las funciones de negocio que se realizan.
- Las relaciones entre estos componentes.
- Los organigramas de la empresa, etc.

La perspectiva de negocios puede expresarse mediante la **modelización de los procesos empresariales** desarrollándolos mediante un modelo de procesos, siguiendo un esquema como el de la figura de la derecha. Los procesos se gestionan mediante **Bussines Process Management (BPM)**.

Entre otros, los elementos a gestionar dentro de un proceso BPM son, entre otros:

- Mapas de procesos.
- Modelización de los procesos.
- Reglas de negocio.
- Modelo conceptual de datos.
- Integración de datos y procesos.
- Descripción de procesos dentro de un marco SOA.
- Diseño de los procesos dentro de aplicaciones distribuidas SOA..
- Mapa de eventos-respuestas.
- Análisis de afinidad y integración de procesos ., etc..

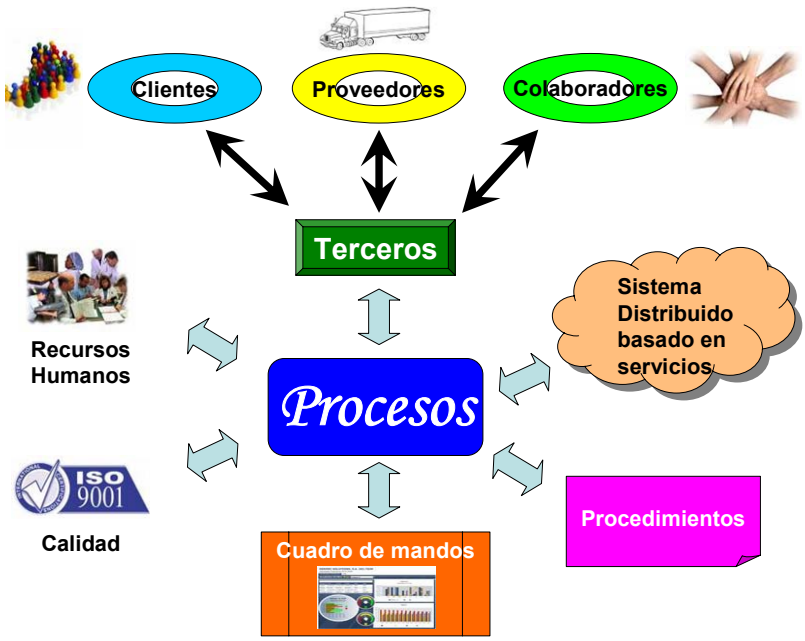


Figura 3. Arquitectura para organizar los procesos de negocio.

3. 2. La Perspectiva de Aplicación (Aplication Perspective).

Define las aplicaciones de la empresa.

Son componentes clásicos de la perspectiva de aplicación:

- Descripción de las aplicaciones existentes.
- Descripción de los servicios (en el sentido que introduciremos más adelante) disponibles, internos o externos, y sobre los que se soportan los procesos de negocio.
- Forma de obtener esos servicios.
- Planes para el desarrollo de nuevas aplicaciones y reingeniería de las antiguas para alinearlas con los objetivos y retos de los negocios.

3. 3. La Perspectiva de la Información (Information Perspective).

Define que necesita saber la organización para funcionar.

Son componentes clásicos de la perspectiva de información:

- La descripción y contenido de los datos.
- **Diccionario de conceptos** donde se explican todos los términos utilizados en la información de la aplicación. Por ejemplo, como se evalúa el seguimiento del presupuesto de ventas o que se entiende por venta real. Observe que esta información pueden ser atributos de más de una entidad o obtenerse como integración de varios de ellos.
- Los modelos de datos y las estructuras de las bases de datos.
- Las políticas de administración de datos.
- Descripción de las diferentes visiones con que esos datos se crean, manipulan y consultan por la organización.
- Procesos de Workflow de datos.

3. 4. La Perspectiva de Gestión (Management Perspective).

Define los condicionamientos de gestión y administración de toda la plataforma distribuida.

Aunque su contenido es muy amplio, son componentes clásicos de la perspectiva de gestión:

- Lugares donde existe administración informática.
- Condicionamientos organizativos.
- Políticas de soporte a usuario.
- Gestión de adquisición de recursos.
- Horarios de disponibilidad.
- Políticas de medición y análisis de rendimientos, etc.

3. 5. La Perspectiva Tecnológica (Techology Perspective).

Propone el software básico, el hardware, las redes y las comunicaciones que soportan el sistema distribuido y por tanto a la organización.

Son componentes clásicos de la perspectiva tecnológica:

- Hardware y software básico de los puestos clientes y de los puestos servidores.
- Estándares adoptados por la organización
- Recursos de impresión.
- Ofimática.
- PDA's y telefonía móvil, etc…

3. 6. Relación entre las perspectivas.

La integración y relación entre las perspectivas de la arquitectura de empresa se muestra en la figura.

Los puntos de entrada simbolizan los inputs externos por la evolución de los objetivos de negocio y la tecnológica.

Cuando el input es a través de la arquitectura de negocio, los inputs funcionales y operativos para las nuevas aplicaciones o los cambios de las actuales se generan desde allí.

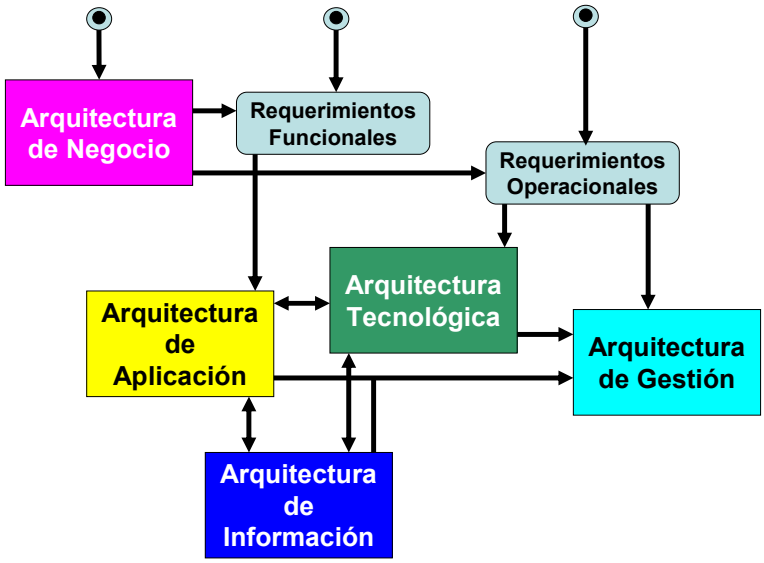


Figura 4. La relación de las perspectivas de la Arquitectura de Empresa.

La flecha entre las arquitecturas de aplicación y de gestión simboliza la instalación de nuevas aplicaciones o cambios de las actuales que han de ser administradas y gestionadas.

Las Arquitecturas de Aplicación Clásicas: Batch y Transaccional

1. Introducción.

Es demasiado frecuente confundir arquitectura distribuida únicamente con soluciones Cliente/Servidor e Internet en entornos gráficos.

Nos olvidamos de las posibilidades de las soluciones Batch y Transaccional de toda la vida y que el boom informático de los 80 y 90 diluyó en los programas monolíticos desde los que surgieron por evolución las soluciones C/S e Internet.

Estas arquitecturas "de siempre" siguen perfectamente vivas y son de gran utilidad en las aplicaciones distribuidas. Hay que conocerlas y usarlas cuando y donde la funcionalidad y la organización del sistema distribuido las necesite.

Vamos a recordarlas.

2. Programa monolítico.

Es difícil buscar un nombre para el programa que "se lo guisa y se lo come" todo. Es decir que todas las funciones que necesita se las implementa el mismo. Dicho de otra forma, no comparte nada ni usa nada que no sea suyo.

Este programa, que voy a llamar monolítico, no debe confundirse con el programa que utiliza servicios estáticamente al estar incorporados al programa en el momento de linkarlo.

Una vez presentado, creo que podemos convenir que no hay nada más que hablar sobre él y olvidarlo.

3. Arquitectura Batch.

Una arquitectura Batch se basa en la ejecución en cadena secuencial de varios programas que cubren una función dentro de la aplicación.

La arquitectura Batch ha de proporcionar:

- **Recursos para definir el flujo de la cadena**, recurso de programación implementado en:
  - **Un lenguaje de definición del flujo.**
  - **Parámetros:**
  - **Filtro y configuración del proceso. Por ejemplo, el periodo de fechas a tratar.**
  - **Control del flujo de la ejecución.**
- **Un mecanismo de petición y ejecución**, denominado por razones históricas por su nombre en los Mainframe, el Remote Job Control (RJC). Aporta varios tipos de recursos:

- Un mecanismo de petición con:
  - Una cola de los procesos pendientes de ejecución con un mecanismo de anotación y de consulta desde el exterior del estado de ejecución. La cola necesita un mecanismo de prioridades y clientes VIP.
  - Un gestor RJE que consultando la cola inicia cada uno de los procesos anotados.
- Parámetros de:
  - Filtro y configuración para fijar las condiciones de cada ejecución. Antes de iniciar la cadena se han informar para esa ejecución en concreto. Todos los procesos de la cadena pueden consultarlos.
  - Control de flujo de la cadena. Los graban los procesos de la cadena en función de los resultados que van obteniendo. Permiten, en particular,
    - Tomar decisiones dentro de la cadena en función de los resultados que se van obteniendo.
    - Enviar información de resultados del proceso al exterior. Así los programas o operadores que han encolado la cadena Batch disponen de información de lo que ha pasado. Por ejemplo, se puede informar al exterior si la cadena ha acabado bien o con error.

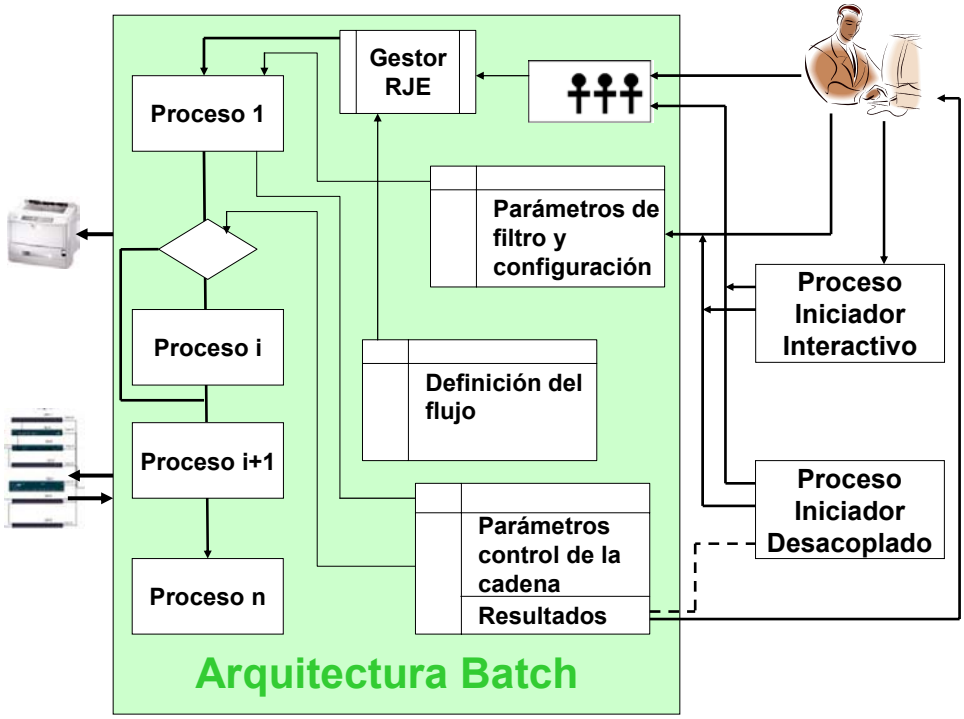


Figura 5. Arquitectura Batch

La utilización del mecanismo de ejecución puede hacerse de varias formas.

Un operador responsable de explotación puede:

- Encolar el trabajo manualmente después de informar los parámetros de filtro y configuración modificando directamente la definición formal de los parámetros. Normalmente utilizará un recurso de tipo editor.
- Utilizar un programa de presentación GUI para entrar y validar los parámetros y que de forma desentendida encola el trabajo. Esta segunda solución tiene dos ventajas claras.

- ❍ El operador puede tener un nivel de formación más bajo.
- ❍ Se filtran de entrada los posibles errores y coherencia de los parámetros.

Otra forma de encolar trabajos puede ser **desacoplada a partir de otros procesos del sistema distribuido**. Por ejemplo, un proceso de aceptación de albaranes en almacenes dispara una cadena Batch de una aplicación centralizada para incorporar los datos a la aplicación corporativa de administración.

Cuando se activa el gestor de RJE, que actúa como un agente (término que si no conoce aprenderemos más tarde) arrancado y vigilando la cola, los trabajos se van lanzado y ejecutando a partir del inicio de la cadena.

Cada uno de los programas de la cadena se configura con los parámetros de ejecución registrados, realiza su trabajo e informa del resultado de su gestión en los parámetros de control de la cadena.

Después de la ejecución de cada paso, se sigue el flujo definido tomando decisiones si es necesario analizando los parámetros de control y/o de configuración.

Durante el desarrollo de la cadena se van actualizando los datos y pueden enviarse listados a impresoras generales o asignadas a usuarios o departamentos.

Al final, los parámetros de control de flujo de resultados quedan disponibles para su consulta al exterior.

Si el proceso batch se ha lanzado automáticamente y el lanzador se ha esperado a obtener los resultados, recibe los parámetros de resultados pactados. Esta ejecución síncrona no es nada recomendable por eso está indicada en trazo discontinúo en la figura.

4. Utilidad de la Arquitectura Batch.

Puede pensarse que esta arquitectura solo es útil en Mainframe. Nada más lejos de la realidad. Bien al contrario es uno de los mecanismos útiles más olvidados y menospreciados en las aplicaciones distribuidas por diseñadores que confunden distribución con interfície gráfica.

Aunque a lo largo de este texto deberemos volver a hablar sobre este tipo de arquitectura conviene dejar ya de entrada constancia de la utilidad del proceso por lotes que supone:

- ❍ Incremento de la productividad por la planificación y el solape de tareas
- ❍ Optimización de recursos distribuyendo la carga sobre ellos uniformemente en el tiempo.
- ❍ Automatización de tareas administrativas repetitivas.
- ❍ Automatización de las tareas que suponen necesidad de altos conocimientos.
- ❍ Disminución de errores por operatorias erróneas.
- ❍ Análisis de incidencias a posteriori por expertos.
- ❍ Localizar en el tiempo las tareas que hay que hacer en momentos determinados.
- ❍ Sincronizar tareas evitando que un operador haya de estar pendiente de ello.
- ❍ Adaptabilidad a la carga de trabajo..
- ❍ Y un largo etcétera.

Todo ello se traduce en:

- Fiabilidad del trabajo.
- Reducción de costes ya que se consigue:
- Hablar de fiabilidad es hablar por si solo de menos costes.
- Optimizar recursos.
- Liberar a los operadores de la vigilancia y el error en procesos largos y repetitivos y que puedan dedicar ese tiempo a otras funciones.

5. La arquitectura transaccional.

La arquitectura transaccional apareció en los primos momentos de la informática comercial por la necesidad de optimizar los recursos de proceso en un momento en que eran un bien escaso.

En la filosofía de Mainframe, cuando muchos usuarios se conectaban en línea, el sistema se colapsaba. Para resolver este problema se creo la arquitectura transaccional.

La idea pasa por separar en un programa los datos del proceso. Por cada instancia del programa arrancada se guarda una copia de los datos, la página de datos en el argot, y solo se asigna un componente de proceso cuando se necesita.

Un componente de presentación, localizado en el entorno del usuario, le está atendiendo mientras estudia y elabora la información de la siguiente entrada.

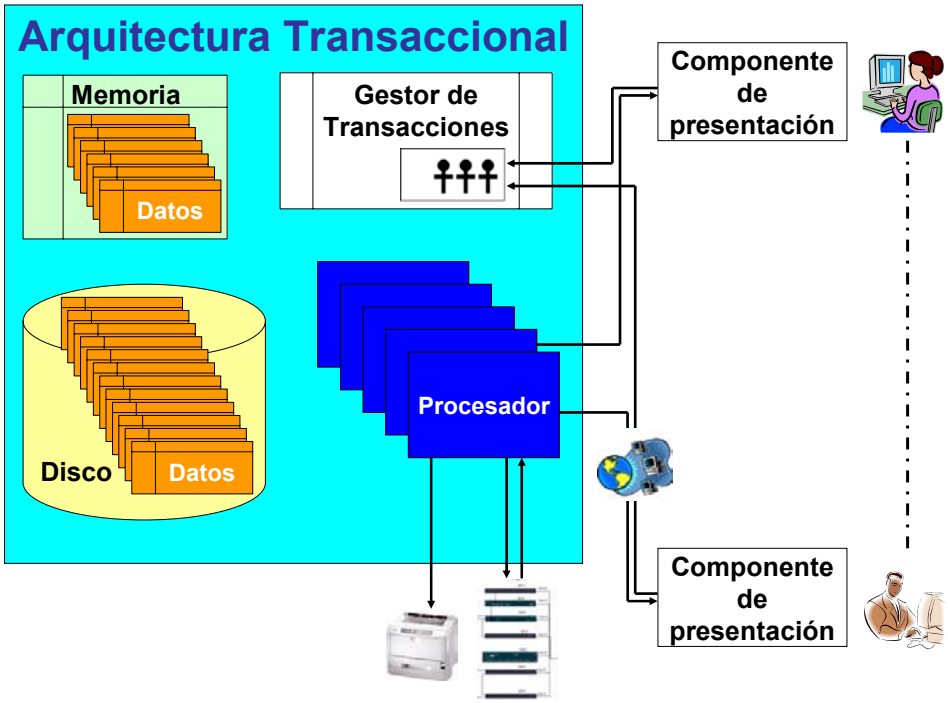


Figura 6. Arquitectura Trafnsaccional

Repasemos la secuencia de trabajo.

1. Cuando el usuario indicia el proceso, el Gestor de Transacciones le asigna una copia de datos y un hilo de proceso.

2. El hilo de proceso prepara la pantalla de presentación y la envía al componente de presentación que pasa a atender al usuario. La página de datos se archiva y el hilo de proceso queda liberado para dar servicio a otros usuarios.
3. El usuario estudia los datos que se le presentan y prepara la siguiente entrada que va a realizar a partir de ellos. El componente de presentación realiza el filtro de errores e incoherencias de datos que se van a registrar tan a fondo como su entorno le permite asegurando al máximo, dentro de sus posibilidades, que la entrada que se va a iniciar tendrá éxito.
4. Cuando el usuario inicia la transacción, el conocido INTRO de toda la vida, el gestor de transacciones recupera la página de datos de ese usuario y le asigna un hilo de proceso que recibe la página de datos del usuario.
5. El hilo de proceso toma el control, realiza el proceso que se le ha pedido y devuelve la respuesta de la transacción al usuario repitiéndose de nuevo los pasos anteriores.

Como opciones adicionales:

- Las peticiones de atención de las transacciones se archivan dentro del monitor de transacciones en una cola para impedir que se pierdan peticiones si el procesador tiene ocupados todos sus hilos.
- Las páginas de datos más frecuentemente utilizadas se guardan en memoria y las menos utilizadas en disco.
- El componente de presentación puede estar trabajando remotamente al otro lado de la plataforma de comunicaciones.
- El componente de presentación puede ir desde una interfície GUI hasta un simple gestor de presentación no inteligente.

Es evidente que se consigue así que la capacidad de servicio del sistema sea muchísimo mayor que si un hilo del procesador quedará permanentemente asignado a un usuario.

Cuando se habla de monitores transaccionales, hay que rendir homenaje histórico al CICS de IBM, el primer gran monitor de transacciones universalmente utilizado.

¿Este sistema le parece obsoleto? Revise el funcionamiento de una WEB convencional. Un usuario se conecta y recibe sobre el navegador la pagina HTML que ha solicitado. La CPU de housing de la WEB guarda los datos de la sesión del usuario y se despreocupa de él hasta que vuelve a entrar una petición de atención de ese usuario momento en que recupera los datos de su sesión y le concede servicio del servidor WEB. ¿Le suena? Bienvenido a un ejemplo actual de arquitectura de funcionamiento transaccional.

La búsqueda de los Orígenes

1. ¿Cuál es el nombre de cada cosa?

Esta pregunta tan simple, tiene para nosotros los informáticos una respuesta muy compleja y en algunos casos casi imposible.

Cuando uno consulta documentación informática de cualquier tipo, se encuentra con un lío terminológico sencillamente increíble. Las mismas cosas se llaman de forma diferente, cosas diferentes se llaman igual y cosas que no tienen nada que ver con la Informática se incluye dentro de la documentación informática como conceptos básicos.

Las razones son muchas y variadas, pero quizás puedan centrarse en tres grupos:

- La necesidad de los departamentos comerciales de presentar siempre nuevos productos aunque sólo sean ampliaciones o maquillajes de productos ya existentes.
- La presencia de grandes compañías cada una con su propia terminología.
- Las consultarías, los investigadores y autores están introduciendo continuamente nuevos términos.... ¡no siempre para conceptos nuevos!

La razón de fondo es nuestra gran suerte. Vd. y yo, amigo lector, estamos en una rama de la ingeniería nueva, con escasos setenta años de vida. Y que se desarrolla a una velocidad vertiginosa. Los ingenieros de caminos ya hacían obras sólidas en la época de los romanos. Si no acuérdese de monumentos tan impresionantes como el Acueducto de Segovia. Su terminología y sus métodos se han ampliado y mejorado considerablemente, pero tienen la consolidación que solo dan los años. Evidentemente este no es nuestro caso.

Hacer una relación exhaustiva y comparativa de términos informáticos no es mi objetivo. Además pienso ese tarea supera mis capacidades.

Iré centrando lo que necesite a medida que avancemos.

Sin embargo, para recordarle la magnitud del problema a que nos enfrentamos y para centrar de entrada algunos conceptos, permítame que empiece con algunos ejemplos de este monumental lío.

2. El lío terminológico.

Personalmente creo que hasta el lío es clasificable. Yo aprecio tres grupos.

2. 1. El lío funcional.

Dentro de la especificación funcional de sistemas y de sus posibles soluciones aparecen conceptos como:

2.1.1. Proceso corporativo.

Es un concepto que hace referencia a que el proceso informático interesa al total de la compañía.

Es una calificación mucho más organizativa que informática.

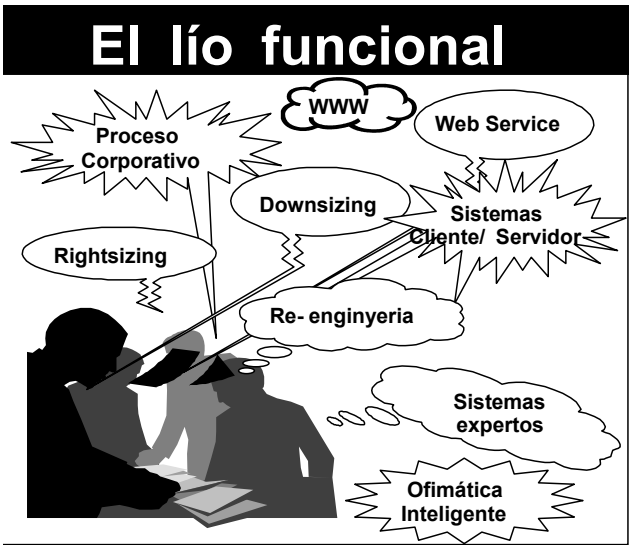


Figura 7. El lío funcional

2.1.2. Proceso departamental.

Hace referencia a que el proceso informático sólo afecta a un departamento.

Como en el caso anterior, es un término más organizativo que informático.

2.1.3. Proceso cooperativo.

Es un concepto que hace referencia a que dos o más elementos colaboran realizando una única tarea repartiéndose el trabajo. Proceso claramente de diseño distribuido y fundamental en nuestro viaje.

2.1.4. Reingeniería.

Es un término muy genérico aplicado a aquellos trabajos informáticos desarrollados para adaptar sistemas antiguos a las nuevas tecnologías y/o necesidades.

2.1.5. Sistemas expertos.

Es una calificación de los sistemas informáticos que hace referencia a que el proceso es capaz de analizar, adquirir experiencia de ese análisis y aplicarla a proponer soluciones delante de un problema de su ámbito.

La parte buena es que la idea es interesantísima. La parte mala es que los departamentos comerciales están vendiendo sistemas expertos que no han pasado del parvulario.

2.1.6. Aplicaciones heredadas.

Son aplicaciones existentes en el momento de diseñar ampliaciones de sistemas de información. Muchas veces se aplica específicamente únicamente a aplicaciones de Mainframe, pero en general, en él deben incluirse cualquier aplicación operativa en ese momento.

Es un término usado con mucha frecuencia y a veces en un inmerecido todo despectivo. No menosprecie la calidad, eficiencia, eficacia y robustez, dada por años y años de uso diario, de muchas de estas aplicaciones.

La mayoría de las veces en que deben ser sustituidas los son por nuevas aplicaciones que tardan meses en funcionar al mismo nivel de servicio y años en disponer de la misma robustez que las sustituidas.

Actúan a modo de precondiciones de los nuevos sistemas.

2.1.7. Downsizing.

Esta es una más de las muchas palabras que se encontrará con la terminación anglosajona *sizing*.

La idea del Downsizing es montar las mismas aplicaciones, con perdida mínima de prestaciones, en máquinas más pequeñas interconectadas y a un menor coste.

Se puso muy de moda en los primeros tiempos comerciales de las arquitecturas C/S en las cuales los departamentos. Comerciales de las empresas con productos en ese campo, vendían la idea de que había que abandonar los grandes ordenadores y sustituirlos por los emergentes sistemas distribuidos C/S.

La idea original del mensaje, todo nuevo, era inviable desde su origen ya que si bien los ordenadores para C/S eran más baratos que los HOST, el coste de desarrollo y sustitución organizativa de las aplicaciones antiguas por las nuevas era inmensamente mayor que ahorro potencial.

Excepto en contadas y muy conocidas excepciones, el Downsizing nunca se ha trabajado en este ámbito general, sino como una forma de reingeniería parcial que afecta a algunos procesos o partes definidas de los sistemas de información.

Demoremos el tema para el capítulo dedicado a reingeniería.

2.1.8. Rightsizing.

Es un concepto que hace referencia a diseñar el mejor sistema informático para que el coste de inversión y organizativo sea mínimo. ¡Casi nada! Si Vd. conoce la fórmula, por favor, publique un libro. Se forrará.

Yo, además de comprarle un ejemplar y de regalarle otro a todos mis amigos informáticos para Reyes, seré su principal apóstol.

Como yo no tengo la fórmula, no podré usar más este término.

2.1.9. Upsizing.

Este término hace referencia a la unión *hacia arriba* de dos sistemas informáticos que habían sido independientes anteriormente. O también incorporar máquinas aisladas a un sistema integrado.

Por ejemplo, su empresa o su cliente tenían una aplicación de facturación en la central administrativa y otra de producción en fábrica y los dos sistemas se quieren hacer trabajar juntos, no por interfases, sino integrados. En ese caso, Vd. se habrá de diseñar un Upsizing.

Este término, enunciado en este marco, es básico en el mundo de las aplicaciones distribuidas ya que el ejemplo que le he puesto es frecuente. Y su solución pasa por arquitecturas distribuidas.

Un Upsizing de este tipo da lugar a problemas de gran complejidad originados por los diseños independientes de las aplicaciones heredades. Piense, por ejemplo, en las diferencias sintácticas y semánticas de todo tipo que puede haber en una entidad como producto que seguro que está en las dos aplicaciones.

Hoy día el Upsizing puede ser tanto de datos como de procesos. Se ha acuñado el termino EAI, integración de procesos corporativos en inglés, para referirse a este proceso. El Upsizing es un tema muy importante y tecnológicamente complejo que trataremos al final de la segunda parte dentro de los temas de Reingeniería aplicando las técnicas de diseño que iremos presentando a lo largo del libro..

2.1.10. Outsourcing.

Es un concepto que hace referencia al desarrollo o a la explotación de circuitos empresariales por terceras empresas especializadas. En términos informáticos es concepto es totalmente paralelo.

Se habla a veces de BPO (Business Process Outsourcing) como el conjunto de proceso de negocio subcontratado que incluyendo tanto los recursos técnicos como los métodos de negocio. En este caso la empresa de servicios suele tener una participación en los beneficios derivados de la gestión subcontratada.

Evidentemente, es un tema totalmente administrativo, de gestión y de costes, que no afecta al proceso informático en si fuera de la obligación de disponer de un buen cuadro de control.

2.1.11. Offshore.

Subcontratación de parte o todos las tareas de desarrollo, mantenimiento i/o soporte a países donde los costes son menores. Se aprovechan los avances espectaculares en conectividad y comunicaciones.

Vamos, más outsourcing, ¿no?

2.1.12. Housing y Hosting.

Son términos alternativos utilizados para referenciar la localización de una WEB en un proveedor de Internet.

Cuando el usuario es el propietario de la WEB se habla de Housing y cuando la aloja en un tercer por algún sistema de Outsourcing se habla de Hosting.

Como observa, el un tema que nos afecta ya que a efectos de diseño solo importa que habrá que localizar el componente, no de quien paga.

2.1.13. SaaS.

El Software como Servicio (**Software as a Service**, **SaaS**) es un modelo de distribución de software en donde la compañía de IT provee el servicio de mantenimiento, operación diaria, y soporte del software usado por el cliente. El cliente tiene el sistema hospedado y subcontratado en la compañía de IT.

Más de lo mismo….

2.1.14. Los términos e-@

E-business, e-b2b, e-commerce, e-market, e-procurement, e-bi (plataforma de negocios inteligentes), e-sourcing, e-movile, e-etc… no son más que nuevos nombres a las funciones de siempre montadas y potenciadas sobre Internet que obedecen muchas veces a razones comerciales. En ellas si que aparece un sistema muy diferenciador: extender la funcionalidad a terceros.

2.1.15. BI, e-BI (Business Intelligence) y EBI (Enterprise Business Intelligence).

Son términos equivalentes que hacen referencia a las capacidades para obtener, a partir de la información archivada en los sistemas de información, conocimientos de clientes, proveedores, personal, mercados, oportunidades de negocio y peligros con el objetivo de obtener mejores beneficios con mayores ventas, menores costes y grado de satisfacción, del personal interno y colaborador, más alto.

Business Intelligence pretende colocar los datos a disposición de toda la empresa extrayéndolos de las aplicaciones donde se generan, pasarlos a un formato estándar, almacenarlos en una base de datos optimizada para conseguir rapidez de acceso en grandes volúmenes de consultas a través de una herramienta de acceso consiguiendo mejorar la gestión y la competitividad de la empresa.

El concepto se parece muchísimo al de Data Warehouse del que hablaremos más delante.

Basado en BI, el Corporate Perfomance Management (CPM) conecta personas y estrategias con los sistemas de gestión para permitir tomar las decisiones orientadas a optimizar el rendimiento global de la organización. Estamos de acuerdo, supongo, que parece una terminología comercial más que de diseño.

2.1.16. Informática inteligente.

Es un término comercial aplicado a paquetes informáticos capaces de dar *altas prestaciones*. Sin comentarios.

2.1.17. Cloud Computing.

Los sistemas en nube, del inglés **cloud computing**, es un paradigma que permite ofrecer servicios a través de Internet.

La nube es una metáfora de Internet, sobre la base de la nube utilizada para representar Internet en los diagramas de red informática como una abstracción de la plataforma que proporciona los servicios,

2. 2. El lío en la arquitectura del sistema.

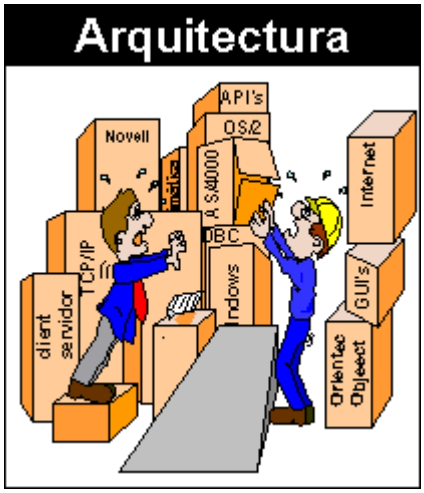


Figura 8. El lío de la arquitectura del sistema

¿Y qué me dice de lío de los nombres y posibilidades de productos y estándares? El caos.

¿Qué producto escoger entre los que hacen cosas parecidas? ¿Y de qué fabricante? (¡Por favor, no me conteste lo fácil!, siempre Microsoft...).

Nos dicen que los productos siguen estándares que encajan entre si. Cualquiera que haya intentado hacer compatible productos de este tipo sabe de los problemas para integrarlos, empezando por la instalación, momento en que unos empiezan ya a hacerse la *puñeta* a los otros.

Sin embargo, al final las cosas acaban más o menos encajando porque los responsables de sistema conocen su trabajo y los jefes de proyecto saben elegir sus productos. Pero todos rezamos para que no nos cambien la versión de alguna pieza antes de que nos hayamos recuperado del esfuerzo de hacer funcionar bien la anterior.

Pero no me malentienda, la situación en este punto es muy positiva. Los estándares han solventado la mayoría de problemas que teníamos y han abierto el camino para ver las cosas de una forma nueva y compatible. Todos nuestros problemas vienen de la carrera desenfrenada por sacar nuevas versiones sin acabar de consolidar las anteriores en un eterno salto hacia adelante.

Este documento que empieza leer no es un tratado de productos y herramientas, sino de diseño de aplicaciones distribuidas. No lo olvide en ningún momento. Es una documentación para diseñadores, no para técnicos de sistemas ni programadores.

Además, supongo que entre que ha empezado a leer este capítulo y acabe el último, van a pasar tantas cosas que si me atrevo a citar demasiados productos actuales, cuando Vd. lea estas líneas pensará que este libro ya está obsoleto.

Al final, las buenas noticias, la generalización de cloud computing puede minimizar este problema.

2. 3. La visión profesional del diseño.

Hoy día los conceptos que se manejan en cualquier diseño informático son básicamente:

- Conectividad.
- Orientación a Objetos
- Cliente/Servidor + Internet
- Ofimática
- Servicios

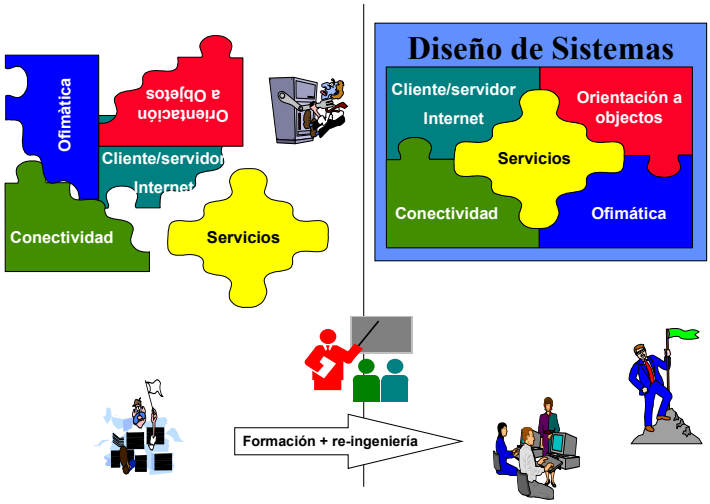


Figura 9. La visión profesional del diseño

Estas disciplinas están lógicamente interrelacionadas y se han de apoyar entre ellas, ofreciendo soluciones globales en las que participan y se ensamblan armónicamente más de una de ellas.

Su llegada ha sido tan rápida que el personal informático se ha sentido en muchos casos desbordado por el sistema y empantanado en un mar de urgencias y documentación, conceptual y de productos.

Para salir del pozo solo hay una solución: formación. Y aprovechando esa formación, crear los nuevos sistemas y aplicaciones y ganar experiencia. Y hacer reingeniería, no siempre substitución, de los sistemas antiguos.

Solo así, y entendiendo que la informática ya hace años que es una rama más de la Ingeniería que obliga a trabajar en equipo reutilizando el trabajo propio y el de los demás, podrá alcanzar el éxito en su trabajo.

Uno de mis objetivos es intentar encajar estos conceptos, además de los clásicos, en el entorno del desarrollo de aplicaciones distribuidas.

3. Cuestiones básicas iniciales.

Para aquellos de Uds. que no tengan todavía alguna idea conceptual clara de que es Cliente/Servidor voy a intentar contestar a tres preguntas básicas:

- ¿Qué es una arquitectura distribuida?
- ¿Cual es su origen?
- ¿Qué se puede hacer con ella?

Para conseguir responder a esta pregunta voy a mirar hacia atrás.

Las aplicaciones informáticas, que tras el diseño y la programación, resuelven las necesidades de una organización apoyándose en una plataforma informática, han evolucionado con el tiempo.

Retrocedamos en el pasado reciente y situémonos en el nacimiento de las arquitecturas distribuidas que inicialmente fueron Cliente/Servidor sobre Sistemas Operativos. Esta visión histórica retrospectiva nos ayudará a empezar a situar el concepto de aplicaciones distribuidas.

4. Un poco de historia.

No se asuste. No pienso *taladrarl*" aquí con hojas y hojas sobre los orígenes, evolución y, por tanto, la historia de la Informática.

Ni sé lo suficiente ni creo que esta historia se pueda contar en unas pocas hojas. La historia de nuestra profesión, aunque corta en tiempo, está llena de acontecimientos. Y exigiría un esfuerzo de investigación y de síntesis que por si sólo ya justificaría un libro de varios tomos.

Lo que necesito es situarle en que momento, y en que circunstancias aparece C/S e Internet y que implicaciones tiene.

Para cansarle todavía menos, voy a ir al punto al que deseo llegar presentando cada etapa de la evolución histórica que me interesa remarcar con una ficha con el formato de la figura.

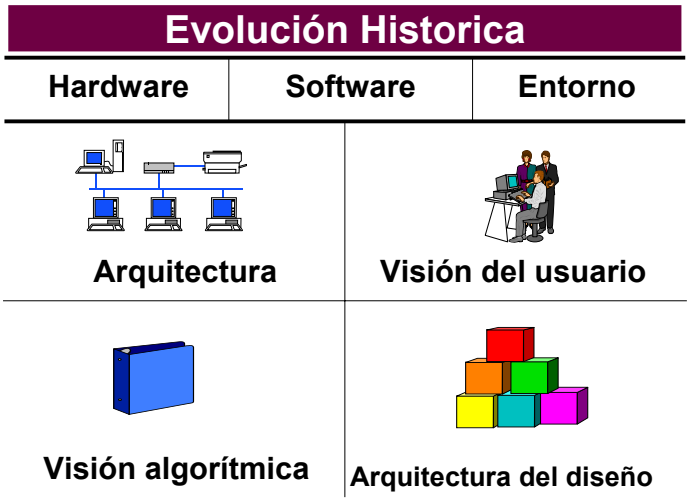


Figura 10. La ficha evolutiva

En los tres cuadros superiores se resumirán las características de hardware, software y entorno organizativo y en los cuatro cuartos:

- Cómo era la arquitectura de hardware.
- Cual era la visión del usuario de la informática.
- La Visión Algorítmica, es decir, como de construían los programas.
- Como se diseñaba la arquitectura de sistema para distribuir las funciones de la aplicación.

Los primeros ordenadores en el mundo de los negocios aparecen a partir de 1958 y abarcan una generación evolutiva que llega hasta mediados de los 60.

Básicamente, es una generación formada por ordenadores con poquísima capacidad de almacenamiento externo, programados artesanalmente.

Poco a poco se desarrolla y consolida el concepto de sistema operativo y la programación es cada vez menos artesanal y más técnica. Aparecen primeros lenguajes simbólicos imperativos las rutinas y la programación estructurada. Cada programa es una isla que inicia y acaba una función.

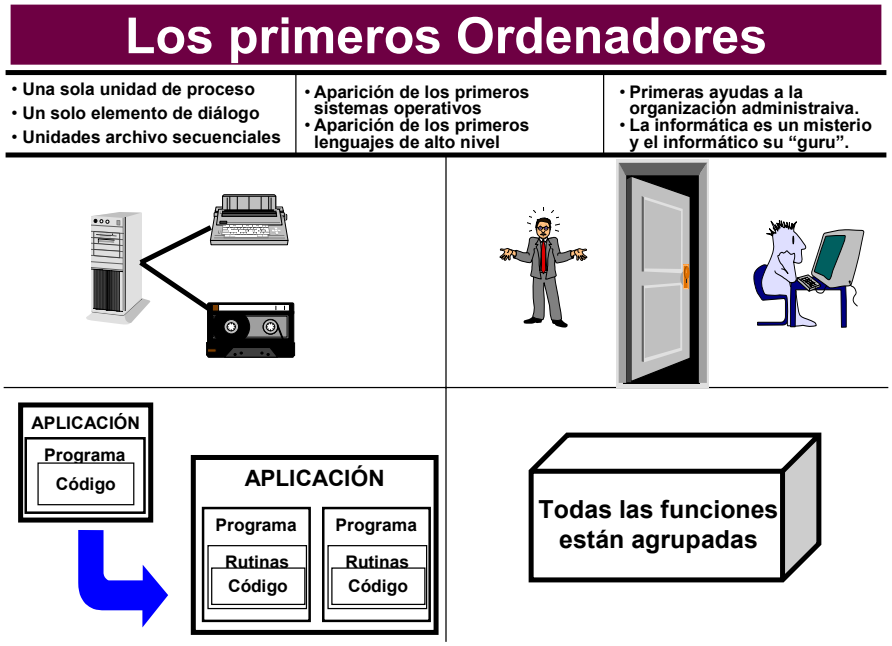


Figura 11. Los primeros ordenadores comerciales.

Entre mediados de los 60 y mediados de los 70 aparecen los **Mainframe**, grandes ordenadores con una sola unidad central y capacidad de atender a más de un usuario simultáneamente. Surge así la idea de HOST u ordenador corporativo. Y aparecen los procesos batch, el teleproceso y los monitores de transacciones.

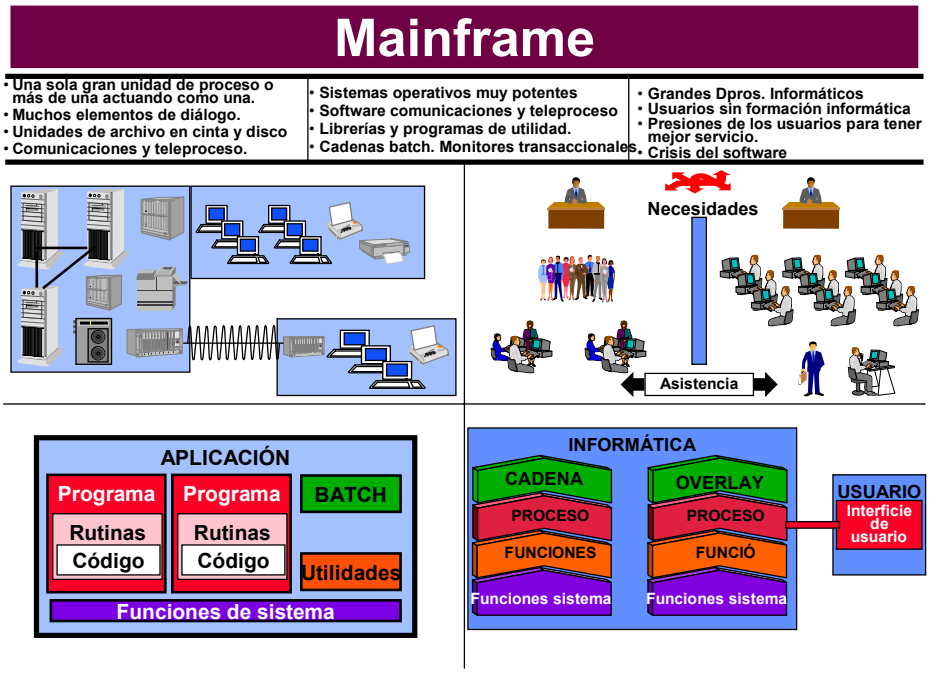


Figura 12. Los Mainframe.

Surgen alrededor del HOST los grandes departamentos informáticos, aislados de los usuarios y que sólo reciben nuevas aplicaciones a través de los elementos directivos. Se crean secciones dentro de los departamentos para dar soporte a los usuarios.

El uso de la informática se democratiza. Empresas más pequeñas que no pueden permitirse un Mainframe, ni tener los grandes departamentos informáticos que conllevan, crean un sector nuevo de mercado. Aparece una generación muy difícil de clasificar que yo referenciaré como de los “Minis” y que emulan un HOST (una sola unidad central que sirve a más de un usuario) a precios competitivos. Para abaratar los costes de personal, se rebaja el nivel de las herramientas de desarrollo y administración lo que disminuye el tamaño de los departamentos de desarrollo y explotación a cambio de menor control. Los primeros intentos de estandarización y la aparición del concepto de paquete estándar permiten a las empresas comenzar a desarrollar externamente.

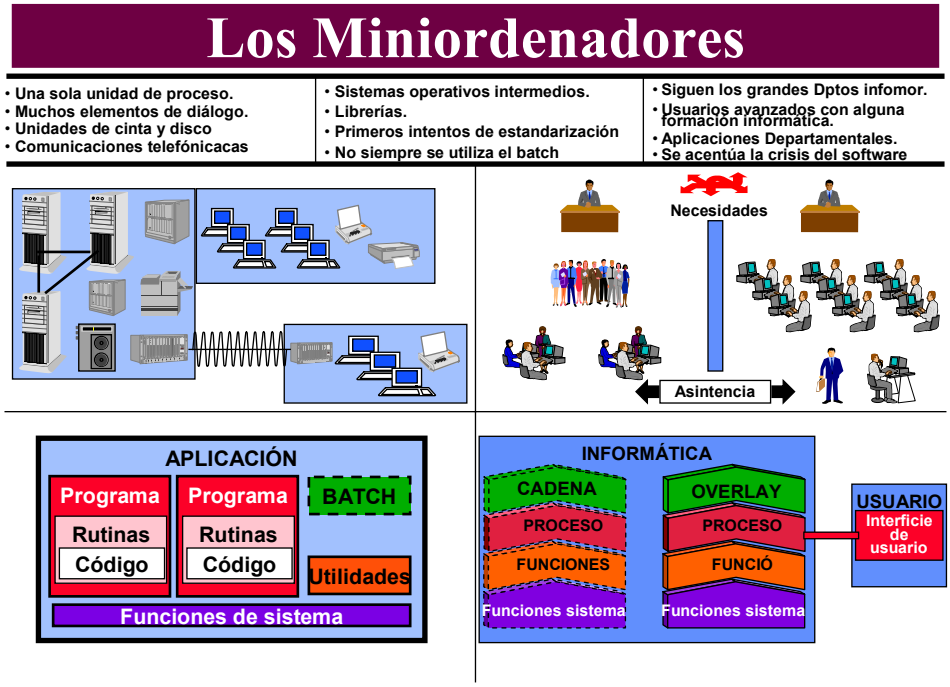


Figura 13. Los Minis.

Se produce en este momento un hecho muy importante: los usuarios toman la iniciativa. Los usuarios y las organizaciones empiezan a exigir nuevas aplicaciones que desbordan las capacidades de los departamentos informáticos atrapados en la dinámica del mantenimiento de lo que ya tienen. Las peticiones de los usuarios desbordan toda previsión y los nuevos proyectos se eternizan. La situación exige una solución rápida y los Minis, menos pesados de programar y que se pueden programar externamente, se empiezan a utilizar para aplicaciones departamentales. Como los departamentos informáticos centrales están agobiados y colapsados, el crecimiento en esta situación es dirigido directamente por los responsables de los departamentos usuario, prácticamente sin intervención de informáticos corporativos. Los precios bajos no ayudan a conseguir buenos programas. Pero la programación resultante, aunque de muy baja calidad, frena de momento el problema y cumple inicialmente su función de descargar tensiones. Las relaciones entre HOST y Minis se establecen a través de ficheros de interfase. Pero la vida incipiente de los Minis llega a su final de forma abrupta.

A principios de los 80 se produce un otro hecho crucial: aparecen los primeros PC’s de IBM. Poca gente cree el ellos hasta el punto que la única persona que apuesta por desarrollar un sistema operativo estándar para la nueva generación de hardware es un desconocido Sr. Bill Gates.

El precio, todavía más bajo para los PC's que para los Mini's, los pseudos estándares creados de facto por Microsoft y la carrera cada vez más alocada de más prestaciones a menores precios hace que esos Minis desaparezcan y los PC's pasen a ocupar la periferia de las aplicaciones informáticas.

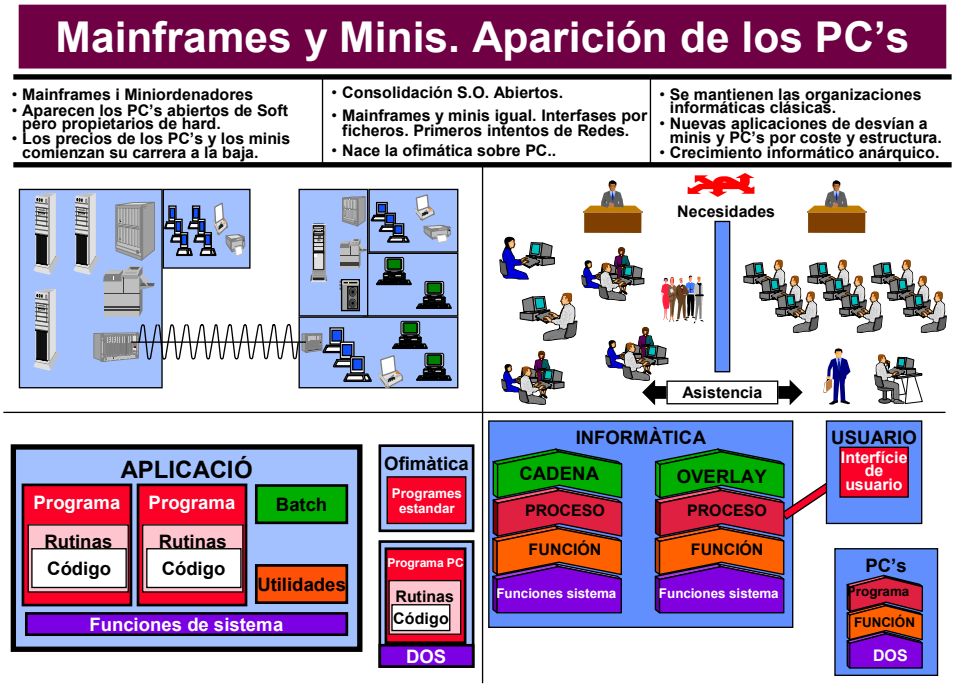


Figura 14. Los Minis y los HOST en coexistencia. Aparecen los PC's.

Los usuarios necesitan colaborar entre ellos y acceder a otros ordenadores donde hay aplicaciones y datos de interés. Surgen así las redes.

La maquina de escribir se queda obsoleta y los programas de edición de textos sobre PC se imponen. Aparecen las hojas de cálculo y todo junto inicia la Ofimática.

Y los entornos gráficos, que hasta es momento solo usaba Apple, se introducen en los productos de PC's.

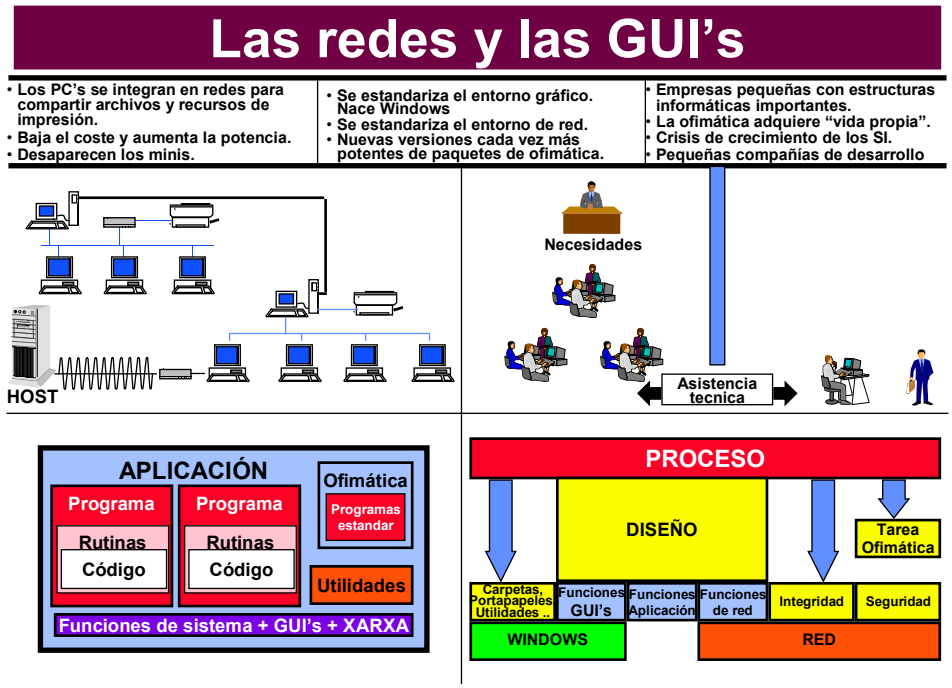


Figura 15. Redes y entornos gráficos.

Las aplicaciones de los PC's van cambiado de objetivos. Cada vez son más importantes y menos periféricas. Aparece un nuevo concepto de diseño basado en aprovechar las ventajas del entorno gráfico de Windows, los recursos de red y las prestaciones ofimáticas.

Y como consecuencia natural de todos ello, los datos corporativos que hasta ese momento se habían relacionado con los PC's a través replicaciones manuales y de ficheros de interfase, pasan a ser necesarios en línea, entidad a entidad y registro a registro. No hay nada preparado para esa situación, nueva en el panorama informático. La reacción es la aparición de las arquitecturas Cliente/Servidor.

5. Nacimiento de la arquitectura C/S.

Analicemos el problema.

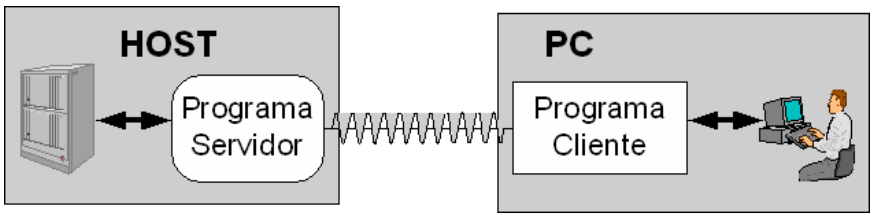


Figura 16. Necesidad de acceso a los datos corporativos desde un HOST.

En el fondo, para acceder a los datos corporativos desde un PC, hay que establecer una comunicación entre dos programas, uno en el lado del PC y el otro en el lado del HOST.

Y para poder comunicarse dos programas es obvio que necesitan intercambiarse mensajes. En esos momentos los modelos de comunicación entre dos programas ya estaban estudiados y se utilizaban ampliamente. Entre todos los modelos posibles, se decide escoger el modelo de comunicación Cliente/Servidor que se caracteriza por la existencia de un mensaje de dos pasos, petición y respuesta.

C/S se adaptaba perfectamente a la necesidad. Para una consulta especializada basta que el programa de la parte del PC haga una petición, el programa de la parte HOST la atienda, localice el dato pedido en la BD corporativa y envíe el mensaje de respuesta.

Llamar al programa de la banda PC que realizaba la petición, **cliente**, y al del HOST, que la atendía, **servidor**, fue una consecuencia inevitable.

Para que los mensajes puedan viajar es necesaria una plataforma de **sistema** con comunicaciones sobre el que un **transportista** pueda intercambiar los mensajes.

Evidentemente la plataforma de comunicaciones puede ser local (más o menos rápida en aquella época) o remota (lenta en ese momento).

Además, el termino Cliente/Servidor tuvo éxito y enraizó comercialmente.

Remarquemos que aunque hablemos de diseño Cliente/Servidor, estamos hablando de diseño de procesos distribuidos a más de un programa con comunicación entre ellos según el modelo Cliente/Servidor.

Y finalmente, para la comunicación se realizó una adaptación de un mecanismo que ya existía en el HOST para lanzar trabajos Batch remotos, el ***Remote Job Control*** (RJE). El mecanismo resultante fue un modelo de comunicación C/S síncrono ***Remote Procedure Call*** (RPC), un clásico que ya le presentaré oficialmente más tarde.

6. Puntos básicos de la nueva arquitectura cliente/servidor.

6. 1. Distribución.

Distribución de datos y procesos entre más de un programa.

6. 2. Cooperación.

Dos programas colaboran para alcanzar un objetivo

6. 3. Especialización.

Una de las partes, la del servidor, es especializada. De este concepto hablaremos más adelante, después de reanudar nuestro viaje histórico.

6. 4. Presencia de más de un ordenador.

Con o sin comunicaciones remotas.

6. 5. Presencia de un transportista.

Necesidad de un transportista de mensajes entre cliente y servidor. **Y el transportista puede fallar dejando el servicio deshabilitado**. Esta situación marca un hecho fundamental en el diseño. Hasta ese momento, si un servicio encapsulado en una rutina linkada con el programa, no respondía ese era el menor de los problemas: probablemente la máquina se había estropeado o el programa estaba mal programado.

En C/S, las dos máquinas que soportan a cliente y servidor pueden estar funcionando perfectamente y fallar el transportista. El cliente deberá analizar que puede hacer para recuperar y/o suplir la caída del servicio. Esta idea es el fundamento de una etapa nueva en el diseño: el **análisis de consistencia**, del que hablaremos profusamente a los largo de este libro.

7. Y la historia continúa….

El sistema operativo, y como veremos más adelante, cambio su concepción inicial ligada a una máquina física y se convierte en un concepto de red. Windows, su Office y sus estándares se imponen masivamente.

A finales de los 90, y para intentar acabar con ese dominio de los sistemas operativos de Microsoft, una serie de compañías intentan buscar una vía alternativa y realizan una fuerte apuesta por Internet, una plataforma que ya existía desde hacia mucho tiempo.

Aparecen las primeras Web's y en un tiempo record el sistema se generalizan. En un primera etapa sin interacción con la plataforma local y con aplicaciones para colocar información de todo tipo en la Red y de venta de productos.

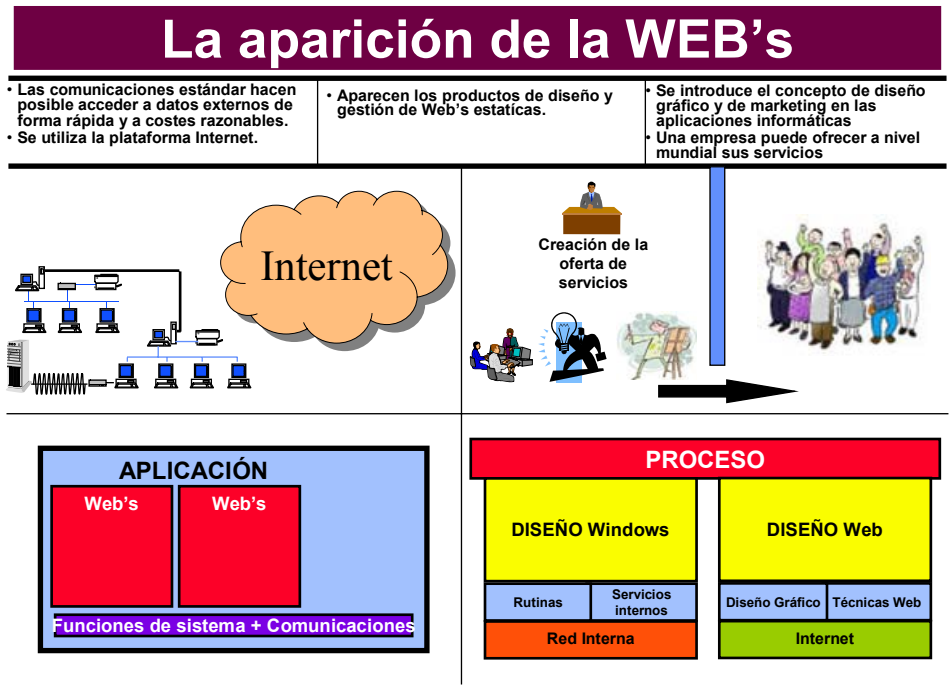


Figura 17. Aparición de las Web's y consolidación de Internet.

Coexisten de forma paralela las aplicaciones convencionales y las diseñadas específicamente para Web.

El cambio es mucho más profundo de lo que en principio parece. La nueva plataforma permite **realizar aplicaciones sin conocer quien ni donde van a ejecutarse**.

Y como el potencial consumidor, al que no se le ve la cara, puede escoger las aplicaciones informáticas y los productos que soportan han de ser vendidas con conceptos de Marketing, una idea hasta ese momento no usada en exceso y que

debe potenciar la imagen gráfica, la facilidad de uso y la renovación continua y atractiva de contenidos.

El cambio supone la interacción entre diseñadores gráficos, artistas al fin y acabo, creadores de ofertas de productos y servicios e informáticos.

La tecnología de diseño de las Web's es claramente cliente/servidor, aunque todo el mundo hable de diseño Internet confundiendo el diseño de contenidos, nuevo, con el diseño tecnológico que es C/S aunque las herramientas sean nuevas,

Obsérvese que en las nuevas aplicaciones Web desaparece el concepto de soporte a usuario ya que no se sabe quien accede y por tanto difícilmente puede dársele soporte.

El nuevo entorno se impone y generaliza tan rápidamente (más adelante profundizaremos en el porqué) que se empieza a actuar con el entorno local y aplicaciones basadas en Internet y Sistemas operativos coexisten e integran. El concepto de servidor se reconvierte definitivamente en servicio y la arquitectura de las aplicaciones distribuidas se define en el perfil de proveedor de servicios. Y si el servicio se suministra desde el propio sistema operativo distribuido o desde Internet en un condicionamiento de diseño o de uso.

Y aparece un razonamiento inevitable. Si las Web's dan servicios, ¿por qué no se pueden usar desde aplicaciones convencionales?

Aparecen así los Servicios WEB que integran todo lo que se puede obtener de Internet como un servicio más para los desarrolladores *convencionales*.

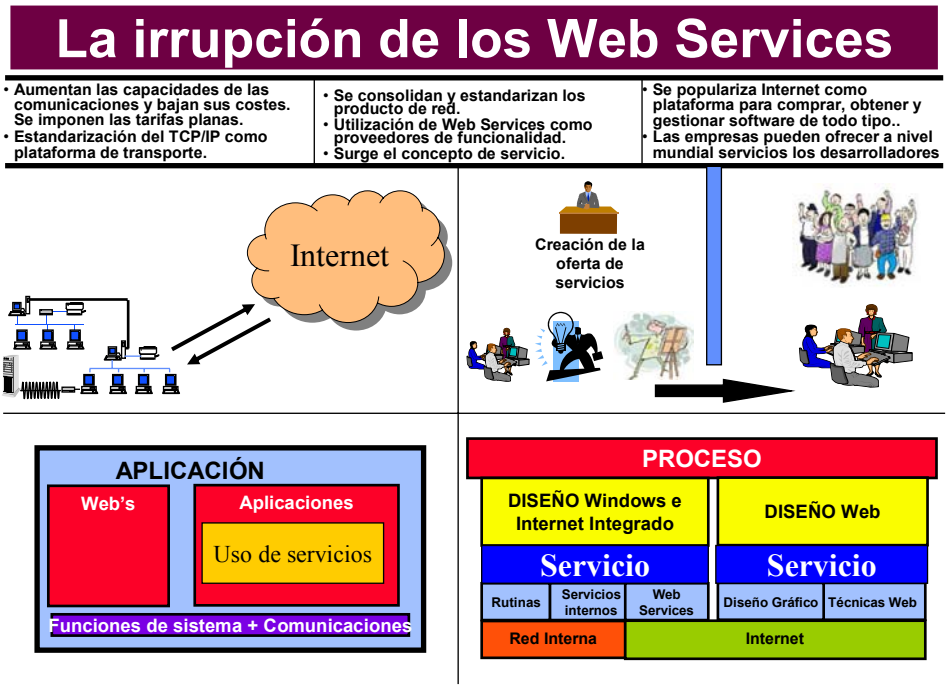


Figura 18. Servicios WEB.

El protocolo TCP/IP base de Internet se impone en todos los sistemas unificando la plataforma de transporte que cada vez el más rápida y barata.

Y, finalmente, el desarrollo descubre que la forma de diseñar sistemas distribuidos con independencia de la plataforma es el concepto de servicio, aplicable tanto a sistemas operativos como a Internet.

Y esa es precisamente la historia de este viaje.

Y valoremos finalmente, el uso de Internet como herramienta de evolución social de la sociedad que ha cambiado, y cambiará mucho más en el futuro, nuestras vidas. Pero este tema es de sociólogos y no de informáticos. Nosotros *solo* diseñamos y construimos las aplicaciones en que se basa esa evolución social.

8. ¿Qué quiere decir especialización?

Hagamos antes de seguir una reflexión sobre el concepto de especialización de un servicio.

La especialización del servicio es un concepto objetivo que no depende del tamaño y la importancia del servicio especializado suministrado por el servidor.

La especialización debe entenderse en tres sentidos:

- Un servicio proporciona una funcionalidad o dato concreto, independientemente de su magnitud o importancia.
- Cliente y servidor se intercambian dos mensajes, uno de petición de servicio y otro de respuesta.
- El diseño y construcción del servidor es independiente y absolutamente transparente al cliente.

Así, tan especializado es un simple servicio de encriptación como una llamada a un procesador de textos para imprimir una factura de formato variable o la petición de los movimientos de una cuenta bancaria a un banco a través de un Servicio WEB.

Para acabar de fijar la idea, observemos el esquema de la figura.

Imaginemos que un proceso S se diseña, con análisis descendente, programación estructurada o cualquier otro método, en el árbol de subprocesos de la figura.

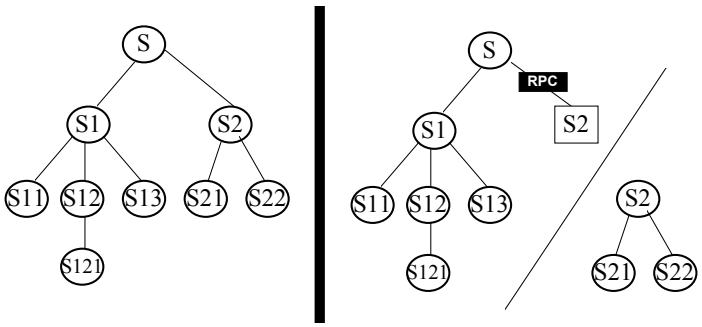


Figura 19. Concepto de especialización

El diseñador decide que S2 será un servidor. Toda la parte del árbol que cuelga de S2 se sustituye por un servidor y se le incluye un modelo de comunicación entre el cliente y el servidor, por ejemplo RPC o Servicio WEB. Todo el subárbol que cuelga de S2 pasa a ser un programa independiente que actuará como servidor y proporcionará la funcionalidad de S2 como un servicio.

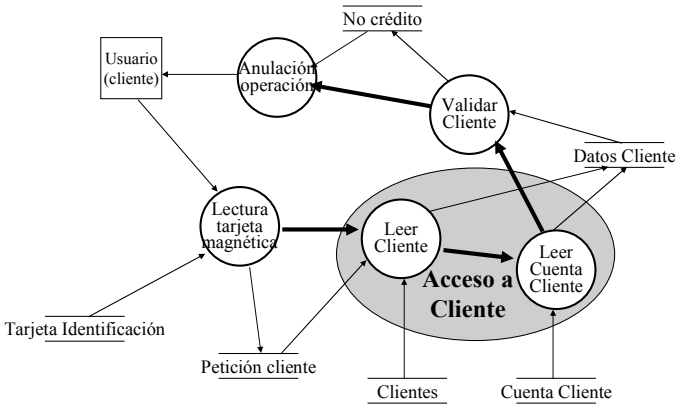


Figura 20. Registro de los datos de un cliente

Esta idea, fundamental en el diseño cliente servidor, es la aplicada en el siguiente ejemplo representado en un diagrama de flujo que propone el proceso a realizar para registrar los datos de un cliente en un proceso de pedidos.

Una vez realizado el funcional, y cuando realiza el diseño de la arquitectura distribuida, el diseñador tiene una precondición para el proceso de acceso a los datos del cliente motivada por la situación de la base de datos.

La BD donde se encuentran los datos está físicamente en una máquina diferente de la que ejecutará el programa cliente. Además el acceso a los datos del cliente supone el acceso a dos tablas, datos básicos del cliente y cuenta contable del cliente. Todo ello le aconseja encapsular la lógica de datos para el acceso al cliente en un servidor que localizará en la parte de la BD.

Una vez tomada esta decisión hará evolucionar el diagrama de flujo de la forma que muestra la figura.

Finalmente, el diseño del servidor de acceso a cliente será el del último diagrama de flujo.

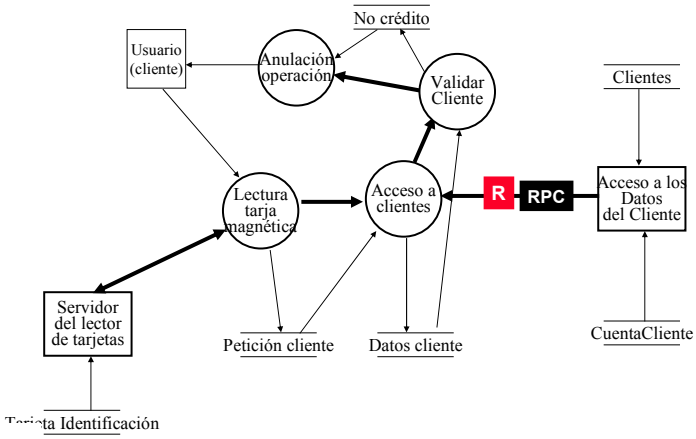


Figura 21. Arquitectura de servidores.

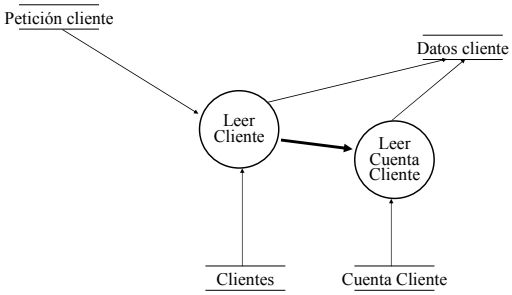


Figura 22. Servidor de acceso a clientes.

Observe que el cliente ha perdido la visión física del servidor y lo único que ve es el servicio. Por esa razón y con la integración de entornos de Sistema Operativo e Internet se ha pasado a hablar de servicio en lugar de servidor como se hacia cuando las aplicaciones distribuidas se montaban en los sistemas operativos y los programas que hacían de servidores eran "tocables" porque estaban diseñados por los creadores de los propios programas clientes.

En Internet los programas servidores, que puede estar seguro que los hay, están escondidos en lugares desconocidos y los han creado y los administran otros, técnicos como Vd. por cierto. Esto hace que hablar de servicio en lugar de servidor sea mucho más apropiado.

9. Primera respuesta a las preguntas básicas.

La visión histórica que hemos hecho permite dar una primera respuesta a las tres preguntas básicas.

9. 1. ¿Qué es una arquitectura distribuida?

Una forma de diseñar sistemas de información basada en la utilización de servicios.

9. 2. ¿Cual es su origen?

Adaptación del diseño a la evolución de los sistemas de información que han tendido a disponer de más de un ordenador cooperando como soporte de procesos y datos de interés general.

9. 3. ¿Qué se puede hacer con ella?

Diseñar aplicaciones más eficientes aprovechando mejor las posibilidades de los recursos informáticos.

10. Servicios.

La evolución supone la confirmación de que el diseño de las aplicaciones distribuidas está basado en **servicios,** independientemente de donde se obtengan y que técnicas se usen para ello.

Los servicios dan datos y procesos especializados proveídos por servidores o agentes.

El diseño distribuido obtendrá y utilizará los servicios en muchísimas ocasiones por técnicas cliente / servidor.

Internet ha potenciado y generalizado enormemente el uso de esos servicios aportando soluciones nuevas, que combinadas con las ya existentes, permiten crear mejores Aplicaciones Distribuidas.

La generalización de los dispositivos móviles, teléfonos y PDA’s, ha abierto otra vía con aplicación a partes del sistema distribuido donde hasta entonces era difícil o imposible proporcionar funcionalidad i/o datos.

Las técnicas de diseño, no de implementación, serán muy parecidas en entornos de Sistema Operativo y de Internet.

11. Estructura básica de un servicio.

Conviene reparar, ya de entrada, que la estructura básica de un servicio serà:

- El servicio en sí, definido por un contrato de especificación y desarrollado y encapsulado de forma análoga a una rutina convencional. Proporciona la funcionalidad y/o los datos. Una de las características de un servicio es que verificará todas las precondiciones, o dicho de otra forma, su contrato no tendrá ninguna precondición..
- La forma de llamarlo (RPC, SOAP, Cola, ODBC, etc..)
- El modo de utilizarlo (asincrona, sincrona i desacopladamente).
- La ficha de enviroment, que permitirá parametrizarlo para adaptar la funcionalidad y ajusta la explotación y la administración.

Más adelante profundizaremos en esta idea bàsica.

12. The Good Old Days.

Antes del nacimiento del proceso distribuido y la irrupción de los PC's e Internet, es decir los tiempos anteriores a Cliente/Servidor y los sistemas abiertos basados fundamentalmente en PC's, cuando los Mainframe dominaban, la vida del informático era fácil

Una vez leí en un libro inglés la expresión ***The Good Old Days*** para referirse a ese periodo, expresión que considero genial, y que me permito citar sin traducir.

La única gran decisión sobre la plataforma y los sistemas informáticos que se tenía que tomar por los directivos del Dpto. de Informática era con que proveedor se "*casaba*". A partir de aquí, a confiar. Y la confianza nunca era defraudada, los sistemas eran robustos y fiables. Caros, eso si. No había posibilidad de compartir entre proveedores nada más que algunos elementos de la periferia.

Además los gerentes de las compañías no cuestionaban las decisiones de los responsables informáticos ya que la referencia de la marca era la garantía.

Los departamentos de Informática eran "*odiados*" pero respetados. Y bien pagados: le garantizo que es cierto, yo lo he vivido. Una factura que los usuarios y las organizaciones se cobraron más tarde.

El software era propietario y limitado, pero cubría sus funciones con solvencia y robustez. Las conexiones de los usuarios locales y remotos eran resueltas por el teleproceso y las trasacciones y la comunicación entre aplicaciones se hacia por traspaso de ficheros. Grandes aplicaciones "de siempre" cubrían los procesos corporativos y nadie se atrevía a discutirlas.

La seguridad, la integridad y la organización eran fáciles y prácticamente impuestas.

La llegada de los PC's, menospreciada por los departamentos de informática clásicos, estaba levantando en alta mar una ola que lo iba a barrer todo.

Perdóneme. No he podido evitar citar la imagen que todo el mundo dibuja de esta situación. Los gigantescos y pacíficos Mainframedocus, dinosaurios herbívoros, pacían y retozaban tranquilamente en la hierba mientras un asteroide de enormes proporciones se acercaba a la Tierra para acabar con su idílica situación.

Sin embargo, la historia acaba de forma diferente a la biológica. El Mainframedocus no desapareció tras el impacto. Después de pasarlo muy mal, evolucionó y se adaptó, disputando el territorio a los pequeños y agresivos carnívoros PC lucha a la que la llegada de Internet ayudo muchísimo. ¡Qué mejores servidores que los potentes y ahora relativamente baratos Mainframe!

13. La vida después de la revolución.

La llegada de los PC's y los sistemas abiertos lo ha cambiado y revolucionado todo.

Aportan libertad de proveedores a cambio de una variedad aterradora de productos y combinaciones que obligan a una difícil selección. Y la oferta cambia continuamente, real y/o comercialmente. Puede elegir entre dos opciones: Bill Gates y Microsoft. Permítame la broma ya sabemos que alguna otra elección tenemos, Linux y AS/400, por ejemplo. Y, ¿que pasará con Google Chrome?

Aparece la estandarización junto con un mundo de nuevas tecnologías que el profesional informático debe aprender. Y es obvio que los jóvenes, que ya han nacido en este entorno han jugado con ventaja. Los "mayores" hemos tenido que reciclarnos. O desaparecer en el intento.

Y ya no hablemos de la revolución de Internet y los servicios, que permiten fabricar en la otra parte del mundo y subcontratarlo todo.

Y los informáticos hemos perdido status. Todo el mundo es un experto informático. Todos los gerentes discuten nuestras decisiones. Nuestros sueldos han bajado y las horas de trabajo han aumentado. Y el número de profesionales o pseudo profesionales ha aumentado espectacularmente.

Cuando veo en el metro anuncios del tipo: aprenda la profesión del futuro, domine la informática, CURSO DE VISUAL BASIC de 120 horas, me sulfuro. Me dan ganas de coger un spray y dibujar un gigantesco graffiti encima. Mis alumnos de la UPC son tontos ¡para qué diablos estudian 5 o 6 años si con 120 horas lo tienen resuelto!

Conceptos Generales de una Arquitectura Distribuida Cliente/Servidor

1. Introducción.

Como veremos más adelante, una arquitectura distribuida participa de los sistemas tradicionales y de cliente / servidor y de Internet.

Como el elemento menos convencional es la parte de arquitectura de esos dos últimos componentes, en primer lugar vamos a presentarlos con detalle y más adelante los interaccionaremos con los sistemas tradicionales.

2. Qué es una Arquitectura Distribuida Cliente/Servidor.

Después de descubrir su origen en la historia, empecemos por centrar que es C/S dentro de una arquitectura de sistemas distribuidos resumiendo las experiencias del capítulo anterior.

Una arquitectura Cliente/Servidor es una forma de diseñar Sistema de Información a partir de **distribuir el trabajo en servicios especializados** basada en la existencia de programas que piden **SERVICIOS**, los **CLIENTES**, a otros programas especializados que los suministran, los **SERVIDORES**. De recoger y transportar los mensajes que se intercambian cliente y servidor se cuida el **TRANSPORTISTA**. Clientes, servidores y transportista funcionan y se apoyan en el conjunto de hardware, sistemas operativos, redes y comunicaciones que constituyen la **PLATAFORMA** o **SISTEMA**.

La arquitectura Cliente/Servidor se convierte en una estrategia de diseño para la distribución de datos y procesos sobre plataformas de sistema en elementos especializados y reutilizables.

Si dos programas actúan de forma coordinada estamos antes un proceso distribuido. Así, hablar de diseño C/S es hablar de diseño de procesos distribuidos en los cuales uno de los lados, el del servidor, está especializado y proporciona servicio a más de un cliente, y en el cual los dos procesos se comunican mediante un dialogo Cliente/Servidor, forma de comunicarse dos programas caracterizada por la existencia de dos pasos:

1. Petición del servicio del cliente al servidor.
2. Respuesta del servidor al cliente con la respuesta al servicio solicitado.

Por ejemplo, si un cliente pide a un servidor una información a través de un servicio de Servicio WEB, la respuesta del servidor al cliente será la información pedida o la causa por la que no ha podido hacerlo, es este último caso, normalmente acompañado de la causa del error. Si un cliente pide a un servidor los datos de un producto, el servidor devolverá una respuesta con esos datos o con un mensaje de que no ha sido capaz de localizarlos. Y así para cualquier servicio de proceso o datos que podamos precisar.

Conviene, en este punto, remarcar también que no es C/S:

- No es una forma de hacer análisis funcional ni de requerimientos.
- No es una metodología de programación.

3. Prerrequisitos de una Arquitectura Distribuida Cliente/Servidor.

Existen tres prerrequisitos básicos en una arquitectura distribuida C/S.

3. 1. Capacidad de ejecutar más de una programa simultáneamente.

Si cliente y servidor no son más que dos programas y se ejecutan al mismo tiempo, es necesaria la existencia de un mecanismo capaz de ejecutarlos simultáneamente. Esta necesidad puede conseguirse de varias formas:

3.1.1.A. Un sistema multitarea.

Capaz de ejecutar sobre una misma maquina más de un programa a la vez. Los recursos del sistema operativo asumen la función de transportista. Una máquina UNIX o Windows proporciona una plataforma adecuada.

Este prerrequisito se da hoy de facto ya que hoy día todos los sistemas operativos habituales son multitarea.

3.1.1.B. Un sistema con más de un ordenador.

Donde cliente y servidor se ejecutan sobre máquinas, y posiblemente sistemas operativos, diferentes. Los recursos de red y comunicaciones proporcionan el transportista para comunicar cliente y servidor. Una red Novel, o una red Windows con nodos UNIX puede ser un buen ejemplo de esta situación.

3.1.1.C. La plataforma Internet.

Donde el servicio se pide a través de la red y se suministra desde plataformas desconocidas para el cliente.

Un buen diseño distribuido hará transparente la existencia de una u otra plataforma.

3. 2. Un sistema de comunicación y de intercambio de mensajes entre programas.

Sobre el que se montará el transportista. Por ejemplo, una plataforma TCP/IP.

3. 3. Potencia de proceso en los clientes en las aplicaciones basadas en sistema operativo.

Con la situación actual de las herramientas y productos de software, en la mayoría de los casos, se hace necesario disponer de máquinas cliente de gran potencia si se trabaja en el modelo basado en sistema operativo. Si esto no disponemos de potencia en la máquina cliente, la situación debe tratarse

como una heterogeneidad de diseño de las que se hablan en el capítulo dedicado a conectividad.

Este punto es bastante menos importante en las aplicaciones basadas en Internet.

También será bueno recordar a que no obliga la adopción de una estrategia distribuida y, en particular, Cliente/Servidor:

- A trabajar con más de un ordenador, a pesar de que casi siempre sea así.
- A comunicaciones remotas aunque cada vez es más habitual.
- A colocar un ordenador por servidor.
- A colocar las máquinas servidoras local y/o remotamente (siempre que la plataforma de comunicaciones esté bien dimensionada técnicamente).
- A separar en diferentes máquinas clientes y servidores.
- A trabajar en paradigma OO, aunque sea altamente recomendable.

4. Características diferenciales del diseño.

La existencia de dos programas actuando normalmente en dos máquinas separadas aporta una característica básica que condiciona todo el diseño: la gestión y recuperación de errores.

En efecto, en un mecanismo *Call-Return* convencional mediante el cual un programa llama a una rutina, linkada en fase de compilación con el programa, la respuesta está garantizada. Es más, si no se recibe, el problema del sistema será de tal magnitud (avería, falta de suministro eléctrico, etc...) que el problema menor que se tendrá es que no se ha recibido esa respuesta esperada.

En un mecanismo Cliente/Servidor la respuesta puede no recibirse sin necesidad de que se esté en una situación extrema: perdida de un mensaje en la red, corte de la comunicación telefónica, etc... Situaciones todas ellas “recuperables”.

Además, si finalmente el servicio no puede “recuperarse”, hay muchas aplicaciones en que hay que continuar dando servicio.

Imaginemos una aplicación que realice la atención del cliente en una tienda de venta de una cadena. La aplicación debe acceder a la Central para confirmar los datos del cliente y su riesgo. Si por cualquier causa la comunicación no es posible, deberemos diseñar la aplicación informática de forma que pueda seguir vendiendo. Si no lo hacemos así, la competencia se frotará las manos ya que le enviaremos nuestros clientes. Por su propia naturaleza, situaciones de este tipo son habituales en aplicaciones distribuidas.

La necesidad de recuperar errores y de añadir funcionalidad para resolver estas situaciones hace aparecer un componente nuevo: el **Diseño de Consistencia**. Es muy importante en C/S basado en Sistemas Operativos y menos en Internet debido a que la plataforma desde donde se suministra el servicio nos es desconocida y los servidores suelen ser máquinas con redundancia ante averías.

De esta nueva etapa de diseño habremos de hablar ampliamente más adelante

La omisión de este componente de diseño en muchas aplicaciones distribuidas ha sido la causa de alguno de los grandes, y muchas veces escondidos, fracasos del mundo C/S principalmente en los basados en Sistemas Operativos.

El tema es más preocupante en las aplicaciones que vayan más allá de procesos de consulta. Si una aplicación de consulta “se cuelga”, el problema más grave que tendremos será acordarnos de los ancestros de alguien, pero en la práctica no habremos perdido nada y rearrancando nuestro sistema o esperando a que los servidores caídos rearranquen podremos seguir trabajando sin más. Es obvio remarcar que en las aplicaciones de modificación de datos o suministro de servicios básicos esta posición no puede adoptarse sin no queremos viajar hacia el oscuro mundo del fracaso.

Por otra parte, dentro del ámbito de administración del sistema, tareas que los sistemas centralizados tenían desde hacia ya tiempo perfectamente resueltas, pasan a no estarlo. Por ejemplo:

Las copias de seguridad, que en un sistema único se hacen con una simple utilidad, pueden pasar a ser una tarea compleja por la necesidad de garantizar la consistencia de datos distribuidos geográficamente en lugares que incluso tienen calendarios laborales y horarios diferentes.
La autentificación única de usuarios en una red mixta UNIX, AS-400, Windows e Internet no es nada trivial.
La distribución de versiones y el control de licencias, y un largo etc...

Los ejemplos son muchos y variados.

Aparece, pues una nueva necesidad.

El diseño ha de proporcionar a los administradores de sistema los recursos que el sistema de administración del entorno distribuido no le proporciona. De ello se ocupará el **Diseño de Administración del Sistema**, componente sin mucho peso en el diseño de un sistema centralizado.

Estos dos componentes “escondidos” se unen al **Diseño de la Arquitectura del Sistema Distribuido** en la que la funcionalidad de la aplicación se distribuirá entre los diferentes componentes, identificando los servidores de cada servicio, estableciendo las reglas de la comunicación entre cliente y servidor, y finalmente, las relaciones entre los servidores.

Estos tres componentes de diseño, que no aparecen en un sistema centralizado, **se habrán de intercalar en la metodología de diseño escogida**. En los capítulos dedicados al diseño se tratarán en profundidad todos estos factores.

Otras características diferenciales del diseño C/S pueden enumerarse en la siguiente lista:

- **Servicios** disponibles, básicamente, en protocolo de petición / respuesta, es decir, cliente / servidor, aunque más adelante veremos que hay otras posibilidades..
- **Especialización** de esos servicios.
- Recursos compartidos.
- **Transparencia** de localización.
- **Independencia** del hardware, sistemas operativos y comunicaciones.
- **Diálogo** basado en mensajes.

- **Escalabilidad** horizontal y vertical.
- **Reusabilidad** de componentes.

5. ¿Qué cualidades conviene buscar en un diseño distribuido?

Para armonizar la arquitectura distribuida el diseñador debe marcarse como objetivo:

5. 1. Trabajar con una imagen de sistema única.

La aplicación debe ser independiente de:

- La infraestructura del sistema.
- La localización de los recursos y servicios sobre el sistema.

5. 2. Adaptabilidad a la organización.

La aplicación deber ser adaptable a:

- Variación de los procesos de negocio.
- Las variaciones de tamaño de los nodos de la organización.
- Al crecimiento modular y escalado.

5. 3. Transportabilidad entre diferentes sistemas

5. 4. Reusabilidad.

Diseñar de forma que se permita la reusabilidad de componentes construidos y la integración de componentes fabricados es una necesidad para conseguir calidad y abaratar costes no solo en un entorno C/S sino en cualquier aplicación informática.

6. Esquema de Niveles.

La existencia de dos programas que colaboran simultáneamente para conseguir la funcionalidad requerida permite establecer el esquema de niveles que se muestra en la figura:

- Un nivel físico que denominaremos **plataforma** del sistema.
- Un nivel de software, constituido por servicios de sistema, donde se apoyan los programas cliente y servidor.
- Un nivel de proceso donde se ejecutan cliente y servidor y donde se intercambian peticiones y respuestas.

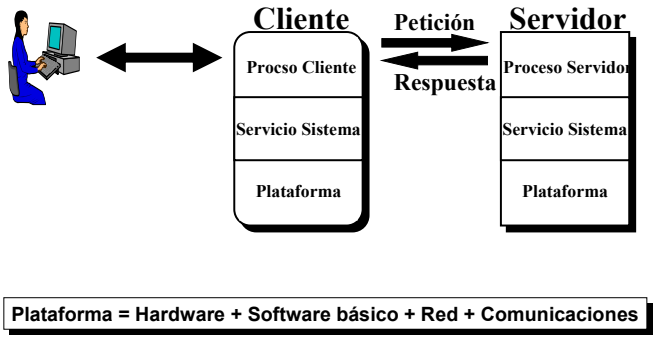


Figura 23. Esquema de niveles.

Desde este esquema inicial y simple los sistemas distribuidos han ido evolucionando hasta modelos más potentes y sofisticados que se irán mostrando más adelante.

Sistema, Transportista y Middleware: Objetivo Proveer el Servicio.

1. Introducción.

En los capítulos anteriores hemos visto que hay cuatro componentes básicos dentro de una arquitectura distribuida: sistema, cliente, servidor y transportista.

El objetivo de un diseño distribuido es conseguir una aplicación transparente al sistema y al transportista. El objetivo será alcanzable en un porcentaje alto de los casos utilizando el concepto de Middleware que presentaremos enseguida.

Sin embargo, en algunas ocasiones el sistema y el transportista introducirán puntos conflictivos que se habrán de tener en cuenta en el diseño como puntos de heterogeneidad.

Dentro de este capítulo se va tratar la infraestructura como elemento que aporta los componentes sistema y conectividad, su encapsulación mediante el Middleware y el aislamiento de los puntos conflictivos como puntos de heterogeneidad en el diseño.

2. Sistema.

Los elementos de hardware, sistemas operativos, redes, comunicaciones, servicios (aportados por ellos), utilidades y paquetes que soportan el/los Sistema/as de Información de una organización constituyen **la Plataforma de Sistema** o simplemente **Sistema**.

La plataforma de sistema de una compañía con años en el mercado es, en general, muy variada y, excepto en contadas ocasiones, una herencia de la evolución histórica de los SI a lo largo del tiempo.

Es muy difícil realizar una clasificación exhaustiva de los sistemas, clasificación que además no aporta demasiado a un diseño que tiene como objetivo director que sus productos sean transparentes a la plataforma.

Sin embargo si existe una clasificación en función de la tipología de los componentes integrados en el sistema:

2. 1. Integración de la cultura de Mainframe y redes de PC´s.

Originada por el crecimiento de los entornos PC alrededor del HOST original. Suele encontrarse en compañías grandes.

En ellas, el Mainframe aporta las aplicaciones y la base de datos corporativas. El PC aporta la facilidad de uso, la potencia y adaptabilidad en el puesto de trabajo, los recursos de ofimática, bajos costes, y modularidad de crecimiento.

2. 2. Redes de PC's.

Suele encontrarse en compañías pequeñas o compañías intermedias surgidas desde finales de los 80.

Las aplicaciones corporativas suelen estar distribuidas por departamentos y la integración suelen ser por interfaces.

Una situación híbrida se encuentra en grandes compañías en las cuales departamentos importantes y, sobre todo, centros de trabajo remotos se han informatizado con redes de PC's. Estas redes soportan aplicaciones departamentales. En este caso hay una conexión estable con el Mainframe, normalmente remota, que constituye un punto de heterogeneidad.

2. 3. Máquinas intermedies (AS/400, UNIX) interconectadas y redes de PC's.

Suelen encontrarse en compañías intermedias surgidas a finales de los 70 y principios de los 80 antes de la llegada de los PC's.

Las aplicaciones corporativas básicas suelen estar en la máquina intermedia pero los departamentos gestionan parte de las aplicaciones sobre sus propias redes u otras máquinas intermedias de menor volumen.

Una situación híbrida se encuentra en grandes compañías en las cuales departamentos importantes se han informatizado con máquinas intermedias. En este caso hay una conexión local estable con el Mainframe. En esta situación puede haber algún punto de heterogeneidad debido a la coexistencia de diferentes sistemas operativos y de soporte de datos.

2. 4. Máquinas intermedias o grandes apoyadas en Web's corporativas.

Suelen encontrarse en compañías medianas y grandes con datos fuertemente centralizados, cultura Mainframe o con gran interacción con sus clientes.

Aquí, el uso de intranets es habitual.

2. 5. Redes de trabajo basadas en conexión a Internet.

Fomentada por compañías aparecidas al principio del siglo XXI, trabajan con PC's con fuerte autonomía en el puesto cliente y conexión permanente con servicios de todo tipo.

Podemos encontrar en este grupo desde compañía cuyo valor añadido es la creatividad hasta proveedores de terceros conectados de forma permanente.

2. 6. PC's individuales conectados y trabajando a través de Internet.

Situación cada vez más habitual. Observe que esta plataforma en realidad una estructura de red remota ya que como centro de soporte de las aplicaciones Web habrá un servidor.

2. 7. PC's conectados a cloud computing.

2. 8. Nuevas combinaciones emergentes con PDA's, telefonía móvil, etc...

3. Infraestructura.

La infraestructura es la concreción física del componente básico sistema. Proporciona los elementos para crear, desarrollar y administrar las aplicaciones.

Hoy día forma un verdadero puzzle que ha de ser integrado para conseguir la arquitectura distribuida necesaria.

La lista de componentes de la infraestructura del sistema puede ser muy extensa. Se incluyen en ella elementos como:

3. 1. Hardware.

- Máquinas y placas de interconexión a red.
- Elementos de comunicaciones.

3. 2. Software

- Protocolos de transporte que interconectan redes y permiten el intercambio de información.
- Sistemas operativos.
- Herramientas de diseño y programación.
- Administradores de red que permiten integrar usuarios y acceder a recursos compartidos.
- Bases de datos y multimedia.
- Productos de ofimática.
- Sistemas de Groupware que permiten la interacción entre personas y el trabajo en grupo.

3. 3. Elementos de seguridad.

3. 4. Internet.

- Web's pasivas de consultas con potencia de búsqueda y localización de esa información.
- Servicios activos.
- Posibilidad de diseñar aplicaciones servidoras que desconocen la plataforma cliente.
- Abasto mundial con acceso rápido y barato.

4. El transportista.

Es el elemento encargado de comunicar cliente y servidor en la solución C/S basada en Sistemas Operativos y de conectar al cliente a la WEB o el proveedor de servicio en la solución Internet.

El cliente prepara el mensaje de petición de un servicio en el formato "pactado" con el servidor y se lo entrega al transportista que localiza la situación del servidor de forma transparente al cliente y le entrega la petición.

El servidor procesa la petición, prepara la respuesta y se la devuelve al transportista que la traslada hasta la posición del cliente que la ha generado entregándole la respuesta.

Así pues, el transportista ha de estar dotado de dos mecanismos:

1. Capacidad para recepción y notificación de mensajes entre programas.
2. Capacidad para transportarlos.

En el marco de actuación del transportista hay que diferenciar tres ámbitos:

4. 1. Local Area Network (LAN)

Una LAN es un conjunto de estaciones (normalmente PC's) interconectadas localmente por un hardware y un software de base que permite intercambiar datos y mensajes y compartir recursos.

4. 2. Wide Area Network (WAN)

Es una terminología que se utiliza para referenciar "puentes" de conexión entre diversas LAN's y/o HOST interconectados no solo no localmente sino que también a grandes distancias.

Una diferencia histórica que ha obligado a diferenciar entre WAN's y LAN's ha sido el ratio de rendimiento. Sin embargo hoy día, y cada vez más, hay menos diferencias entre ellas; las posibilidades de las WAN's están aumentando espectacularmente y derivándose hacia plataformas TCP/IP, fundamento de Internet.

La evolución tecnológica está llevando a estos dos términos al desuso.

4. 3. Internet

Ámbito con vocación de integración global independiente de la plataforma en el cual no hay diferencias de funcionamiento entre local y remoto. La única diferencia real es la velocidad de conexión.

4. 4. Cloud computing

Accediendo vía Internet, se ignora totalmente que hay por detrás.

La presencia de comunicaciones rápidas o lentas en el modelo C/S basado en Sistemas Operativos o la velocidad de conexión del entorno Internet han de ser tenidas en cuenta por el diseñador ya que los procesos no se pueden diseñar igual si el transportista es lento. El tema no es trivial en Internet donde la respuesta de un Servicio WEB puede depender el momento del día o del éxito del servicio.

Habitualmente el factor más afectado por las diferencias de rendimiento del transportista es la gestión de datos. Si la aplicación debe hacer una función que necesita lanzar un conjunto de peticiones de SQL es probable que en local pueda hacerlas de forma convencional. En una WAN, hacerlo así puede llevar a un tiempo de respuesta fuera de los límites tolerables y obligar a diseñar la función usando un procedimiento catalogado en la base de datos.

A efectos de diseño la presencia de un punto en la red global donde existe un transportista lento exige establecer un precondición al diseño. En el método de diseño

que se propondrá diremos que existe un **punto de heterogeneidad** debida a la presencia de un transportista lento.

Observemos que el concepto de transportista lento vas más allá de la velocidad de comunicación física. Se debe tomar en el sentido de tiempo medio para obtener la respuesta.

5. La imagen ideal del sistema y los puntos de heterogeneidad.

Para conseguir que la aplicación sea transparente al sistema y al transportista sería muy interesante que los programas, y por tanto el diseñador, pudieran ver el sistema como una caja negra a efectos de desarrollo y administración del sistema. De esta forma los programas podrían realizarse sin tener en cuenta las diferencias de la plataforma de sistema.

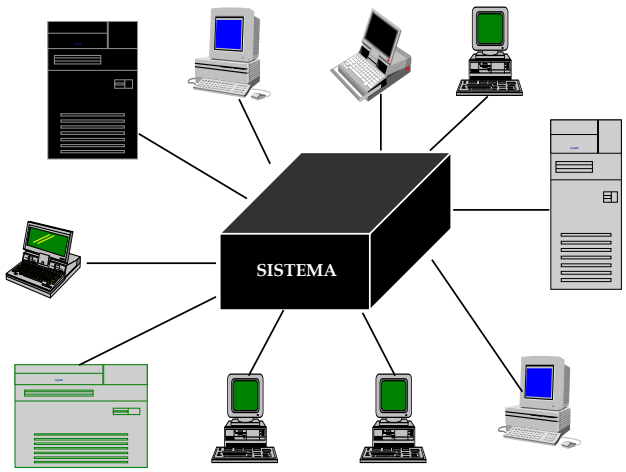


Figura 24. Imagen ideal del Sistema Distribuido

Esta imagen ideal del sistema no debe confundir al diseñador. Pensar que ha vuelto a los orígenes de la Informática, y que su aplicación ya puede ser tratada como si trabajase sobre un único ordenador sería un gravísimo error. Sigue existiendo la posibilidad que no llegue la respuesta del servidor y por tanto la necesidad de prever un análisis de consistencia.

Esta situación idílica está hoy día muy cercana a la realidad y permite mirar la plataforma de sistema y el transportista como una gran **red global interoperable**.

Sin embargo, en la práctica continúan existiendo puntos en los que existe alguna diferencia que no cumple esta condición de interoperatividad.

Al hablar del transportista ha aparecido una de las más habituales: la existencia de puntos entre los cuales existe un transportista lento o congestionado. Rápidamente pueden pensarse otras causas de heterogeneidad como la presencia de bases de datos sin interfaces ODBC o ADO, protocolos de transporte no estandarizados, etc. Se dice en este caso que los sistemas están **interconectados** en contraposición a **interoperables**.

Se hace necesario identificar estos puntos potencialmente problemáticos en el momento del análisis. Se introduce así el concepto de punto de heterogeneidad.

Un **punto de heterogeneidad** es un lugar o situación donde el sistema no se comporta de forma homogénea. Actuarán como precondición al diseño de la arquitectura distribuida del sistema

Conviene realizar una tipificación de los puntos de heterogeneidad que permita establecer una metodología de tratamiento. No se pretende aquí establecer una

relación exhaustiva que, debido a la gran variedad de causas que pueden originarlos, debe quedar en cada caso en manos del diseñador. Podemos establecer algunos a modo de ejemplo:

Punto de heterogeneidad	Efecto	Condicionamiento
Transportista lento o necesidad de levantar la línea	Posibilidad de tiempos de respuesta lentos	Minimizar el volumen de tráfico de datos potenciando mecanismos como servidores de datos en la banda de la BD o los procedimientos catalogados. Prever y controlar los costes de comunicaciones en fase de explotación de la aplicación.
Servicio Congestionado	Demora y dispersión en el tiempo de servicio	Flexibilizar la necesidad de un tiempo de respuesta uniforme. Puede obligar a colocar Sistemas de Amortiguación del tiempo de respuesta basados en trabajar en paralelo con la aplicación y la petición del servicio. Mas adelante veremos que la utilización de mecanismos asíncronos de comunicación C/S facilitará el trabajo.
Middleware heterogéneo (ver en el punto siguiente el concepto de Middleware)	Problemas para conseguir transparencia	Necesidad de encapsular las causas de la heterogeneidad
Máquinas cliente lentas	Poca capacidad de proceso en el lado cliente	Aplicaciones con muchos servidores o necesidad de cambiar las máquinas clientes.
BD sin SQL, o ODBC o ADO	No transportabilidad de la lógica de datos	Encapsular los accesos a las BD si se quiere independencia del motor. Si es viable económicamente cambiar a un motor con BD o adquirir drivers ODBC o ADO para el motor actual.
Transportista no estándar	Dependencia de la plataforma de transporte	Simular un protocolo estándar sobre este transportista o cambiarlo por otro estándar (quizás la mejor solución si se puede hacer). Puede encontrarse en procesos de informática industrial.
Limitación de tiempo el coste de conexión	Disponibilidad limitada de parte de los servicios.	Replicación de servicios en local y almacenamiento de resultados para su transmisión posterior.
Presencia de usuarios móviles o aislados.	No hay posibilidad de disponibilidad de datos en línea de forma eficiente.	Necesidad de replicación e interfaces convencionales a través de intercambio de ficheros planos.
Funciones especificas del sistema operativo (por ejemplo, la gestión de la hora)	Es servicio no es común	Encapsular y especializar el servicio (hablaremos en la segunda parte de cómo hacerlo)
No existencia de Middleware para la gestión de distribución de datos horizontal (1)	Graves problemas para poder gestionar los datos como una unidad lógica y para poder garantizar la coherencia.	Desarrollar software de Middleware para encapsular la heterogeneidad. También puede pensarse en renunciar a la distribución horizontal o llevar un sistema de replicación.

(1) Una entidad está distribuida en más de un esquema lógico; por ejemplo, los clientes de Girona en Girona y los de Barcelona en Barcelona.

Recuerde que los puntos de heterogeneidad deberán detectarse y relacionarse antes de empezar el diseño tecnológico del sistema distribuido.

6. Middleware.

En las primeras aplicaciones distribuidas por C/S, todos los servicios proporcionados por los servidores habían de ser programados a medida. Servicios como impresión, acceso a BD, traspaso de ficheros, etc. que hoy son habituales, debían ser desarrollados por las propias aplicaciones que luego iban a usarlos.

Al finalizar esta **primera generación** de aplicaciones C/S se hacia evidente que este camino debía andarse de otra manera. Los servicios "habituales" podían ser desarrollados de forma estándar y incluidos en todas las aplicaciones como software "prefabricado". Surgen así los estándares de impresión aportados por la red, ODBC, OLE, CORBA, etc... Y una **segunda generación** de aplicaciones C/S, todavía anterior a la explosión de Internet, basadas en su utilización como componentes ya construidos a los que se delega filtrar la heterogeneidad del sistema. Surge así la idea del **Middleware**.

Se conoce como Middleware el conjunto de servicios que permiten distribuir datos y procesos a través de un sistema multitarea, una red local, una red remota o Internet. Estos servicios se catalogan en dos grandes grupos:

- Servicios de desarrollo.
- Servicios de administración.

El objetivo final del Middleware es conseguir transparencia:

- Proporcionando la capacidad de pedir y recibir servicios de forma transparente al sistema.
- Liberando a los diseñadores y a los administradores de sistema de los problemas creados por la complejidad del sistema distribuido.

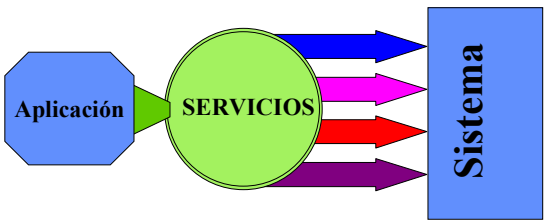


Figura 25. La visión del sistema a través del Middleware

La capa de Middleware está formada por un conjunto heterogéneo de software que puede ir desde servicios muy simples, como un servidor de encriptación, a programas completos como puede ser Word utilizado como un servidor para imprimir facturas en una aplicación distribuida, y servicios Internet conseguidos a través de Servicio WEB.

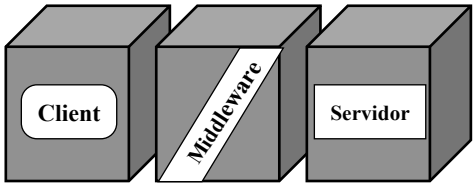


Figura 26. Esquema de bloques de una aplicación C/S.

De los cuatro componentes básicos de un entorno distribuido, el cliente, el servidor, el sistema y el transportista, el Middleware asume y hace transparentes los dos que dependen de la infraestructura: sistema y transportista. Así, en las aplicaciones modernas, puede trabajarse con un esquema de bloques con **únicamente tres componentes: Cliente, Servidor y Middleware** donde el sistema y el transportista se contemplan como servicios del Middleware.

En la mayoría de los casos, el Middleware es capaz de independizar incluso el hecho de que la implementación C/S está basada en Sistemas Operativos o Internet.

La idea de Middleware ha sido reforzada y potenciada con la utilización masiva de Internet. Los estándares y la forma de trabajar de Internet y Intranet han motivado la aparición de un **entorno C/S universal** sobre el que se ha construido **la tercera generación** de aplicaciones distribuidas.

La idea del Middleware es simple y a la vez potente. Volveremos a hablar de él en el capitulo dedicado a implementación e integración.

Existe una clasificación de los sistemas distribuidos en función de donde se colocan Cliente, Middleware y Servidores que se explicará más adelante.

7. Esquema de niveles ampliado.

Cuando existen servidores disponibles dentro del sistema la aplicación queda liberada del trabajo de programarlos. En este caso el esquema de niveles queda modificado tal como se muestra en la figura. El dialogo entre cliente y servidor se establece, no a nivel de programa de aplicación, sino a nivel de sistema.

La aplicación llama al Middleware a través de una rutina local, linkada o no en el programa. Esta rutina local localiza la situación del servicio dentro del sistema distribuido de forma transparente al programa. El servidor atiende el servicio y responde de forma análoga.

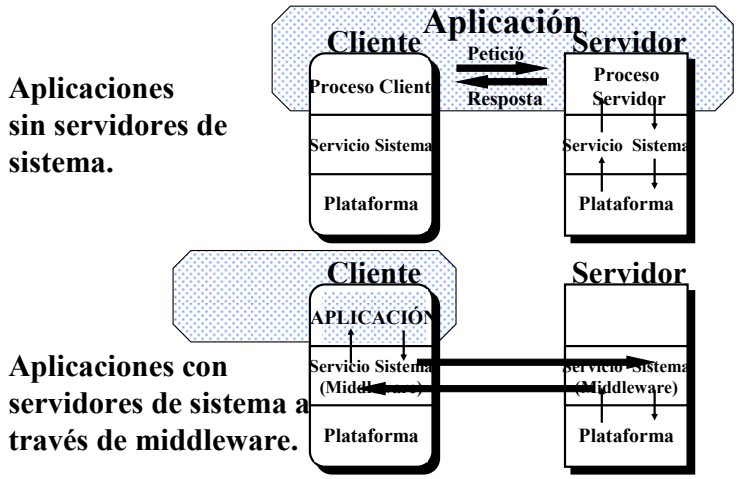


Figura 27. Esquema de niveles ampliado.

La “no visión” del servidor por el programa puede llevar a ignorar en el diseño que la comunicación es Cliente/Servidor. Ello constituye un grave error ya que el sistema continua siéndolo y por tanto todas las características del diseño distribuido perduran (puntos de heterogeneidad, localización, administración, etc.). El único, e importantísimo cambio, es que el servicio ya está programado y probado.

Note que, en este esquema ampliado, no existe diferencia entre la plataforma C/S basada en Sistemas Operativos e Internet.

8. Concepto de localización.

Un buen diseño distribuido se realiza estableciendo servidores que permiten una utilización totalmente transparente de la infraestructura del sistema.

Sin embargo, cuando la aplicación entre en explotación los servicios se habrán de situar sobre los elementos físicos de la infraestructura. Este proceso se denomina **localización** de los servicios.

La localización, y buena gestión, de los servicios durante la etapa de explotación del sistema, será responsabilidad del Administrador del Sistema que trabajará a partir de las directrices establecidas por los diseñadores y de los estándares de la instalación.

El diseñador debe establecer, como parte de su proyecto, el plan de localización de los servidores sobre los recursos que le proporciona la infraestructura. La inercia natural del proyecto sitúa esta etapa como última del diseño. Sin embargo la experiencia demuestra esa tendencia puede llevar a graves errores en el diseño motivados por la no consideración de puntos de heterogeneidad detectados cuando se realiza la localización. Por esa razón, en la metodología de diseño propuesta la localización se adelantará a la etapa de decisión de la arquitectura distribuida.

La localización del sistema se establece normalmente “en frío”, de mutuo acuerdo entre el Diseñador y el Administrador, estableciendo donde y cuando arranca cada elemento. En el caso en que los servidores se arranquen cuando se arranca cada el elemento del sistema donde están localizados, se dice que la **localización es estática**.

En un sistema de una mínima complejidad, el Administrador ya necesitará poder activar y desactivar servicios “en caliente” una vez arrancado el sistema. Es lo que se conoce como **localización dinámica**. Justifican esta necesidad razones como la gestión de averías, atender a puntas de carga, etc.

La localización de un sistema distribuido sobre una plataforma es un conjunto de localización estática y dinámica.

El diseñador deberá dotar a la aplicación de cualidades como replicación y paralelismo (posibilidad de arrancar más de una copia del servidor simultáneamente), tolerancia a fallos, etc. que permitan al Administrador realizar una gestión estática y dinámica del sistema cómoda, eficaz y barata.

Como se verá más adelante, estas prestaciones, que impactan directamente en el diseño de Administración del Sistema, también deben tenerse en cuenta durante el resto de etapas de un diseño distribuido.

La idea de localización se corresponde con la vista de despliegue de las metodologías basadas en OO.

La localización es un concepto de gran importancia en las soluciones C/S basadas en Sistemas Operativos y de muchísima menos incidencia en soluciones Internet.

Puede hablarse de varios tipos de localización:

8. 1. Localización estática.

La localización del servicio se conoce perfectamente y el cliente se limita a tomarla de un fichero de configuración.

Es una solución muy rígida que siempre que se pueda debe ser evitada.

8. 2. Localización dinámica.

Cuando arranca o la primera vez que necesita el servicio, el cliente ha de pedir al Middleware la localización del servicio.

Obsérvese que esta gestión del diseño puede cubrir diferentes situaciones:

- Una localización estática pero que puede cambiar de posición de tanto en tanto.
- Localizaciones alternativas.
- Paralelismo.
- Localizaciones variables del servicio.
- Y un amplio etc.

Un recurso de localización dinámica es, por ejemplo, JNDI de las arquitecturas J2EE.

8. 3. Localización con conectividad cambiante.

Las características de conectividad del transportista cambian con el movimiento del cliente. Suele ser una característica de los servicios móviles que se traduce en variaciones de velocidad y fiabilidad de la comunicación.

Suele producir en un punto de heterogeneidad y en la necesidad de afinar la consistencia.

8. 4. Servicios con componentes móviles.

Los componentes de software que proporcionan el servicio pueden cambiar de localización en el periodo en que un cliente usa el servicio. Influye en el diseño del servicio, no en su uso.

Se da en dispositivos móviles.

Servidores y Clientes

1. Introducción.

Clientes y servidores son las estrellas de nuestro espectáculo y por tanto serán tratados ampliamente más adelante.

Sin embargo, para poder tratar otros temas que se verán a continuación, conviene recordar y fijar las ideas básicas sobre su naturaleza y comportamiento.

Vamos a hacerlo así en este corto capitulo dejando para más adelante su tratamiento y diseño en profundidad.

2. Servidores.

El servidor es el elemento especializado que proporciona la funcionalidad o los datos, es decir el **servicio:** datos compartidos, información activa como es estado de un pedido, recursos físicos compartidos, servicios de impresión, ficheros, funciones comunes, funciones centralizadas, etc.

Su **posición** es **reactiva**, es decir, está inactivo esperando la petición del servicio, la recibe, la procesa, proporciona la respuesta y vuelve a quedar inactivo a la espera de la llegada de la siguiente petición.

Su modo de **ejecución** es **continuo**, es decir se arranca al encender el sistema y está ejecutándose hasta que se desconecta. Sin embargo, veremos más adelante que esto no es siempre radicalmente así. Los servidores pueden arrancarse y pararse fuera de la conexión y la desconexión del sistema. Pero, en cualquier caso, el servidor está arrancado con filosofía de ejecución continua, atendiendo las peticiones o inactivo si no las tiene.

El arranque de un servidor es normalmente **estático**, es decir, el servidor se arranca como parte del arranque general de la plataforma donde está localizado o del subsistema donde se integra. La especificación de los parámetros del arranque están detallados en los ficheros *.INI o XML configuración de arranque o en tablas de configuración de la base de datos.

El servidor puede ser arrancado también de forma **dinámica** por el cliente que primero necesite de él. El tema será tratado más adelante cuando hayamos desarrollado otros conceptos. Citemos aquí, y solo a modo de ejemplo, dos casos en que es habitual:

- Para evitar cargar en demasía máquinas servidoras, servidores que se usan muy esporádicamente, en frecuencia o periodicidad, los arranca el primer cliente que los necesita.
- Si un servidor va muy cargado de trabajo, la aplicación puede decidir arrancar otra copia de ese mismo servidor que, trabajando en paralelo con la anterior, aumentará la capacidad de trabajo del servicio que encapsula el servidor.

En cualquier caso, le recuerdo, que el servidor estará siempre obligado a esconder los detalles de su implementación a sus clientes.

Los servidores forman parte del Middleware básico o se apoyan de forma notable en él. Unos servidores pueden llamar a otros, pero hay que evitar llamadas anárquicas entre ellos. Este tema, que llamaremos **arquitectura de servidores**, será tratado en profundidad más adelante dentro de la segunda parte dedicada al diseño.

Destaquemos finalmente dos características fundamentales:

- **Reusabilidad.**
- **Relocalización (*re-alocability*)**, o capacidad del servidor para localizarse en cualquier punto de la plataforma de sistema.

3. Tipos de servidores.

Sería imposible, y además un grave error, intentar clasificar los servidores. Hay tantos tipos como servicios sea Vd. capaz de imaginar.

Sin embargo, hay algunos servidores que por historia, tradición y reusabilidad se encuentran muy frecuentemente en sistemas distribuidos.

Estos servidores han pasado de ser artesanales, en los primeros tiempos de C/S, a estar estandarizados (la mayoría de ellos) e integrados en el Middleware.

Vamos a enumerar algunos de los más habituales.

3. 1. Servidores de ficheros.

Son servidores que reciben como petición de servicio transmitir en cualquiera de los dos sentidos un fichero completo. En la inmensa mayoría de los casos la organización del fichero es secuencial y muy frecuentemente su contenido es texto, muchas veces en formato XML zipeado.

3. 2. Servidores de datos.

Son servidores que reciben petición de acceso a un registro o campos de una tabla de BD. Hablaremos siempre de BD ya que hoy no se concibe un sistema distribuido sin una BD relacional como soporte de datos.

Un ejemplo de tarea realizada por un servidor de datos es el acceso a una fila de una tabla de clientes a partir del código de cliente utilizando SQL.

Son servidores muy estándares situados en la parte de la BD, servidos por un gestor de base de datos y que se compran fabricados. Prácticamente todos responden hoy día a los estándares de SQL.

3. 3. Servidores de lógica de datos.

Son servidores que encapsulan procesos de datos que suponen varios accesos a la BD para cubrir una funcionalidad determinada.

Los servicios de lógica de datos se construyen a partir de los servidores de datos. Por ejemplo, podemos definir un servicio de datos que, además de los datos básicos de cliente, añada el riesgo concedido y el consumido, datos que pueden estar en otras dos tablas.

Veamos otro ejemplo. Imagine que Vd. tiene una entidad producto repartida en dos bases de datos físicas y quiere hacer un servidor que libere a los clientes de tener que acceder y coordinar el acceso a las bases de datos siempre que necesite algo de productos. Encapsulará la gestión de la entidad producto en un servidor de lógica de datos.

Otro ejemplo habitual es encapsular el acceso a una entidad que tenga sus atributos en más de una tabla.

Estos servidores están muy ligados a cada aplicación y/o instalación. Evidentemente son programados a medida y se apoyan, como ya hemos dicho, sobre los servidores de datos de la plataforma C/S.

Son servidores menospreciados en muchos diseños y que vamos a revindicar aquí.

3. 4. Servidores de Impresión.

Son los servidores encargados de encapsular el uso de las impresoras, recurso compartido por excelencia.

Hoy día, se utilizan los servidores y servicios de impresión de la plataforma, mayoritariamente basados en las soluciones de red.

3. 5. Servidores de Programas.

Son servidores donde se guardan y localizan los programas. Fundamentales en C/S basado en Sistemas Operativos, menos importante en Internet.

Su utilización no es intensiva, reservándose su uso para arranques desde menús o en caliente de unos programas desde otros, como puede ser un arranque dinámico de un servidor.

Sin embargo la presencia de servidores de programas es fundamental en una plataforma distribuida. Lo verá en seguida con la siguiente disyuntiva: ¿Donde coloco los programas, en el servidor de la red o en las máquinas cliente?

Si los coloca en el servidor de red, la facilidad de administración será máxima, pero cuando se estropee el servidor, envíe a su organización a la cafetería. Si por el contrario decide replicar los programas en todas las máquinas cliente, la dificultad de administración será mucho mayor.

Y el tema no es trivial: el trabajo administrativo para controlar y distribuir nuevas versiones, por ejemplo, cambia mucho en una u otra situación.

No hay una respuesta mágica al tema de la distribución de los programas en servidores de programas. La mayoría de las veces los criterios a utilizar son de sentido común: minimizar riesgos con la mínima administración. Por ejemplo, si tiene un paquete de ofimática muy completo, puede colocar los productos habitúales en cada máquina cliente y el resto en servidores. No

coloque en las máquinas cliente programas que no pueden hacer su trabajo sin recursos que están ligados a máquinas servidoras: localícelos sobre servidores de programas ligados a esas máquinas. Prime, si puede, la presencia de pocos servidores de programas, aparte de las propias máquinas cliente.

En una aplicación que no funcione sin la base de datos, el servidor de aplicaciones se localiza habitualmente sobre la misma máquina que el servidor de la base de datos.

3. 6. Servidores de recursos compartidos.

Pueden existir recursos de hardware localizados en una de las máquinas de la plataforma que deberán ser compartidos y no dispongan de Middleware. Su gestión se encapsulará siempre en un servidor.

Por ejemplo, imagine que tiene una placa de encriptación por hardware localizada en una máquina de la red. Cree un servidor de recurso compartido y localícelo en la misma máquina donde esté el recurso.

3. 7. Servidores de Fecha y Hora.

Que todos los nodos del sistema distribuido tengan la misma fecha y, sobre todo, la misma hora, decalajes horarios aparte, es en muchos casos crucial y no puede dejarse al azar.

Por esa razón existen los servidores de fecha y hora donde todos los clientes y servidores pueden sincronizarse con una fecha y hora de referencia.

La llamada a estos servidores suele hacerse al inicio de las sesiones de trabajo.

3. 8. Servidores de impresos.

Proporcionan los formularios de los documentos de la aplicación.

La ventaja de gestionar los impresos como servicios son grandes:

- Su formato está controlado.
- Gestión del mutidioma.
- Facilidad de modificación.
- Etc.

3. 9. Servidores ofimática.

Hay gran cantidad de servicios que pueden ser proporcionados por los paquetes de ofimática: impresión de documentos con todas las facilidades de un procesador de textos, gráficos, las agendas como elemento de planificación, etc.

Cuando utilice uno de los programas del entorno ofimático para cubrir estas funciones estará utilizándolo como un servidor de alto nivel.

Todas estas funciones pueden pedirse a los paquetes a través de API's o llamadas parametrizadas. Conozca a fondo las posibilidades y no caiga en

diseños tan erróneos como desarrollar un subsistema para preparar e imprimir facturas de formato variable y multidioma. Eso se lo hace Word con una posibilidades a las que Vd. necesitaría dedicar mucho tiempo y esfuerzo sólo para acercarse.

En muchas ocasiones los servidores de impresos se resuelven con procesadores de textos.

Hoy día, estos servidores son mayoritariamente productos del Sr. Gates.

3. 10. Servidores de transacciones.

Encapsulan transacciones distribuidas de datos.

Los proporcionan los Monitores Transaccionales.

3. 11. Servidores de Objetos.

En plataformas como modelos de objetos distribuidos (J2EE, .NET, CORBA, DCOM) proporcionan esos objetos.

Es un tipo de servidor en la línea de los servidores de programas y que está incluido dentro de las prestaciones básicas de cualquier modelo de objetos distribuidos.

Si utiliza una de estas plataformas, los aprenderá dentro del proceso de formación.

Muchos de los servidores de Objetos son hoy día Servicios WEB

3. 12. Servidores de Groupware.

Dan acceso a servicios de trabajo en grupo (Groupware), tema que se introducirá más adelante. Incluimos en estos servicios el correo electrónico muchas veces utilizado como plataforma de intercambio de mensajes en el sistema distribuido.

Son funcionalmente parecidos a los de ofimática.

3. 13. Servidores de Multimedia.

Encapsulan la gestión de imágenes y sonido.

Hay mucha costumbre de linkar las funciones multimedia con los programas y pocas veces se encapsulan en servidores.

3. 14. Servicios WEB.

Proporcionan, con cobertura mundial, servicios de proceso y de datos a través de Internet. Los trataremos con detalle más adelante.

3. 15. Servidores de Aplicación.

Finalmente, denominaremos servidores de aplicación a todos aquellos creados para encapsular servicios que no tiene sentido nada más que en el contexto de las aplicaciones especializadas.

4. Clientes.

El cliente es el elemento que pide y usa el servicio que proporciona la funcionalidad o el dato.

Su postura en **proactiva**, es decir está trabajando y cuando lo necesita llama al servidor. Soporta la lógica de las aplicaciones.

Realiza, entre otras funciones, el diálogo con los usuarios a través de interfícies gráficas de usuario y la gestión y recuperación de errores.

El cliente arranca, realiza su trabajo y acaba. Utiliza a los servidores y debe evitar llamadas directas a otros clientes.

Elementos Físicos de la Infraestructura y la Conectividad de interés en el Diseño Distribuido.

1. Introducción.

En este capítulo vamos a tratar elementos *físicos* de la infraestructura, las redes y las comunicaciones.

Pero no se asuste. Este capítulo no es la relación exhaustiva de elementos de hardware, redes y comunicaciones. Está planteado con un doble objetivo. Colaborar a esa cultura general que sobre estos elementos físicos debe tener cualquier diseñador de aplicaciones distribuidas y presentar como pueden aprovecharse de ellos en algunos momentos de su trabajo.

En relación con el segundo punto, quiero hacerle ver una idea fundamental. El diseño de aplicaciones distribuidas es una especialización donde algunas veces, la decisión de un diseñador delante de una situación no es de software si no de infraestructura o conectividad.

Déjeme adelantarle el tema con un par de ejemplos.

Imagínese que tiene una red de local de velocidad lenta y ha de diseñar una aplicación con un gran tráfico de datos. Cuando realiza la adaptación de la aplicación a la plataforma de que dispone (que como veremos más adelante es un paso del diseño distribuido) se da cuenta que no puede atacar los servidores de datos directamente desde los programas por transacciones SQL porque los tiempos de respuesta no serían válidos. Tiene dos opciones: complicar el diseño o ver si el mercado de la conectividad le permite aumenta a precio razonable la velocidad de la red. Si la segunda opción es válida y sus costes son razonables, elíjala siempre.

Imagine que prevé que el arranque de una aplicación le aumentará tanto la ocupación de un servidor que se le quedará corto y además necesita garantizar la fiabilidad del servicio. A lo mejor su solución pasa por colocar un clúster (si no conoce que es lo encontrará explicado más adelante).

Con este objetivo, en este capitulo vamos a tratas diferentes temas en esa línea.

Una advertencia previa. En cuanto empiece a leer este capítulo se dará cuenta de que este tema no es mi especialidad. El capítulo está lleno de simplificaciones, realizadas para presentar las ideas, que asustarían a un especialista. No pretendo escribir de lo que no sé. Sólo pretendo presentar algunos puntos de estos temas que creo importantes para un diseñador distribuido. Si quiere información de mayor calidad, por favor, busque documentación específica.

2. Redes.

2. 1. ¿Qué es una red?

> Una red es un conjunto de ordenadores conectados entre si que permite compartir recursos y intercambiar información entre ellos. El conjunto es interoperable.

Las razones para hacerlo son claras:

- Compartir programas y ficheros.
- Compartir recursos de impresión.
- Compartir líneas de comunicación remota.
- Compartir conexiones de Internet.
- Disponer de correo electrónico.
- Creación de grupos de trabajo.
- Administración de usuarios y seguridad.
- Etc.

No le voy a cansar con más detalles de algo que seguro Vd. lector conoce perfectamente. Este apartado está dedicado a remarcar aquello que, a mi juicio, se ha de tener en cuenta en los diseños distribuidos.

2. 2. Componentes de una red.

2.2.1. Estación de trabajo.

Ordenador, normalmente un PC *normal*, que actúa como cliente de la red y desde donde trabajan los usuarios. Es el lugar natural donde normalmente se ejecutan los programas cliente.

El diseñador debe conocer la potencia de los ordenadores cliente ya que condiciona de forma importante la capacidad de los programas clientes.

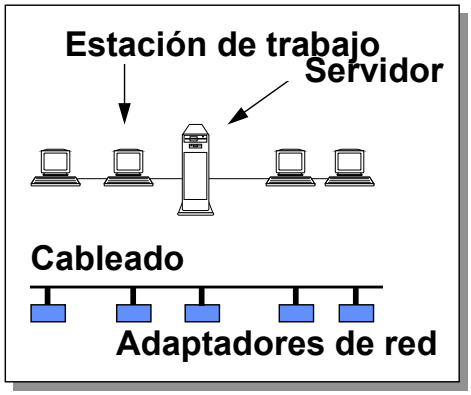


Figura 28. Componentes de una red

2.2.2. Servidores de red.

Ordenadores, normalmente potentes y especializados, donde se ejecuta la parte importante del servicio operativo de red y desde donde se proporcionan la mayoría de servicios de red. Si no forma parte del esquema de niveles lógicos del sistema distribuido (de los que hablaremos más tarde) han de ser ignorados por el diseño. Si forman parte de esos niveles lógicos, son lugares naturales para situar servidores.

Si se utilizan para situar servidores que no son de red, el diseñador deberá asegurarse de que tienen suficiente potencia para dar el tiempo de servicio necesario.

2.2.3. Adaptadores de red.

Elementos de enlace entre los componentes físicos de la red. Su relación con el diseño es la velocidad de tráfico de red que consiguen.

2.2.4. Cableado y conexiones inalámbricas.

Medio físico de comunicación y transmisión entre los componentes de una red. Sin ninguna influencia en el diseño.

Además existen redes inalámbricas cada vez más usadas que permiten la utilización inmediata de terminales móviles en local.

2. 3. Servicios de interés para aplicaciones distribuidas.

Las redes proporcionan, entre otros servicios:

- Servidores de ficheros, datos e impresión.
- Gestión centralizada y unificada de los recursos comunicaciones remotas de la plataforma para disponer de:
  - Acceso unificado a Internet.
  - Servidores de FTP para implementar servidores de ficheros remotos e interoperables de forma transparente a los sistemas conectados.
  - Integración de puntos de red remotos, de forma interoperable si las redes locales y remotas son homogéneas y cuasi-interoperable si no los son. Aunque en este último caso se ha de estudiar caso por caso.
- En redes homogéneas, servicios unificados e integrados de gestión y autentificación de usuarios y gestión de seguridad de acceso a recursos compartidos.
- Comunicaciones externas con terceros.

Todos estos servicios de comunicación remota **se pueden escalar** de forma transparente con tantas líneas y/o recursos de comunicación como se necesiten. Esta prestación es importantísima.

La gran importancia de todo ello es que se puede utilizar en la implementación de aplicaciones distribuidas de forma transparente a través de servicios de Middleware, la mayoría de ellos servicios directos de red como componentes del DSM (*Distribuited System Management*) del que hablaremos más tarde.

2. 4. Topología de la red.

La topología de una red hace referencia a la forma física con la que se configura.

La tipología de la red no tiene influencia en el diseño, si en el funcionamiento. Por eso, su importancia en el diseño es indirecta. Puede que por razones de servicio, un departamento de la empresa tenga horario de trabajo diferente de otros. Y para evitar que toda la red este funcionando todo el tipo, se decide montar una tipología de red que permitan trabajar a cada departamento por su cuenta. Ello supone, evidentemente, la existencia de como mínimo un servidor por departamento y una conexión entre ellos. Y, evidentemente que las aplicaciones tengan en cuenta que no siempre toda la red está disponible.

Esta es, precisamente la influencia indirecta, que suele estar presente en la etapa de localización de servidores.

2. 5. Protocolos.

Son las reglas y procedimientos que utilizan las redes para establecer la comunicación en los nodos. Son el soporte del transportista. Sin importancia en el diseño ya que están embebidos en el Middleware.

2. 6. Interconexión de redes.

Las redes locales se pueden ampliar simplemente añadiendo más estaciones. Sin embargo la evolución natural del crecimiento de la red se encontrará más tarde o más temprano con tres situaciones.

- El crecimiento de estaciones habrá llegado a saturar el servidor y hay que colocar otro servidor o bien hay que colocar estaciones muy alejadas (aunque en local) del servidor.
- La necesidad de integrar dos redes independientes.
- La necesidad de segregar una parte de la red para conseguir que una parte de las estaciones pueda trabajar con mayor velocidad de red (por ejemplo, porque en parte de los clientes se instala una aplicación de diseño gráfico).

Son necesarios, pues, **adaptadores de red** que permitan organizar el conjunto con las prestaciones que se necesitan. Consulte documentación específica si desea mayor información sobre este tema:

Para un diseñador la concreción física de este tema no tiene ninguna importancia; para eso están los especialistas en conectividad. Es sólo cultura general. Debe conocer que existen estas posibilidades y en caso de necesitarlas y para asegurarse de que su aplicación funcionará correctamente, consultar a expertos en conectividad

La evolución de las redes, normalmente en un proceso doble de expansión e integración puede seguir diferentes caminos según la situación inicial y las necesidades de crecimiento de cada compañía.

3. Clústeres.

3. 1. ¿Qué es un Clúster?

> Un clúster es un conjunto de dos o más ordenadores acoplados en red que comparten un subsistema de discos, muchas veces en replicación por espejo, y un software que trabajan como un sistema único garantizando el funcionamiento normal del sistema aunque falle uno de los ordenadores.

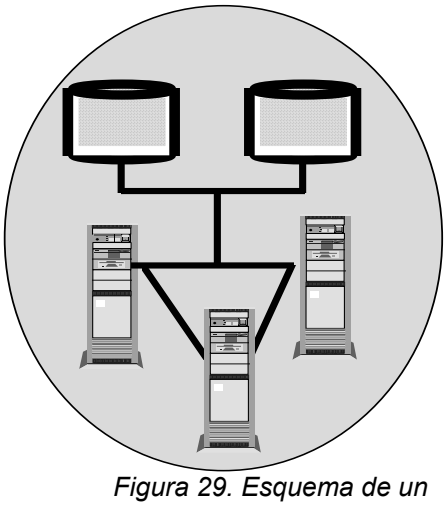


Figura 29. Esquema de un Clúster

Los elementos de un clúster son:

- Los nodos de proceso.

- La red de interconexión.
- El subsistema de discos compartidos.

Se comparte el acceso a disco y los recursos de administración de datos pero cada nodo dispone de su propia memoria y sistema operativo.

Los nodos de proceso pueden ser ordenadores independientes u ordenadores con multiprocesadores o ambas cosas simultáneamente.

El software del clúster reparte la carga por los nodos de forma transparente. Así el efecto exterior es de un único ordenador que aporta una altísima fiabilidad de funcionamiento.

Como los ordenadores integrados en el clúster están integrados entre ellos por una red, la velocidad de esa conexión de red es fundamental en el rendimiento del clúster. No es lo mismo una conexión entre los ordenadores del clúster por cable Ethernet que por fibra óptica.

El clúster aparece para los programadores y administradores como un sistema único.

Cuando hay una caída de algún elemento del clúster la reconfiguración es automática y de forma transparente a los programas y usuarios. Otros elementos del clúster absorben las funciones que estaban cubriendo los elementos que han fallado. El proceso no puede durar más de 15 o 30 segundos. Los usuarios nada más notan una bajada del rendimiento si el nodo va muy cargado.

El software del clúster lanza mensaje de avería al centro de administración del sistema distribuido y la reparación se puede hacer normalmente el caliente.

Nótese que recuperar un proceso que ha quedado interrumpido a medias en el momento del fallo y redirigirlo a otros procesadores y discos de forma transparente no es ninguna obviedad. No es extraño que los buenos fabricantes de clusters guarden el secreto de su software con gran cuidado y que los buenos clusters no sean baratos.

No debe confundirse un clúster con ordenadores multiprocesador y multidisco, que si bien aportan mayor fiabilidad de servicio que servidores convencionales no son clusters. Como rápidamente se aprecia, el secreto está en el software.

3. 2. Ventajas de los clusters.

3.2.1. Alto nivel de disponibilidad de datos y aplicaciones.

Superior en muchos casos al 99%. Los **tolerantes a fallos** alcanzan fiabilidad del 99,9999% y los de **tolerancia total** (*full tolerance*) ofrecen virtualmente un 100%. Pagando claro está.

3.2.2. Aumentos de rendimiento.

Cuantas más CPU's, mejor rendimiento. Sin embargo, el aumento no es lineal:

- de 1 a 2, entre 1,6 y 1,8.
- de 1 a 4, entre 2,5 y 3

- de 1 a 8, entre 5 y 6.

De cualquier forma, no debe olvidarse que el rendimiento está siempre ligado a la aplicación, al sistema de base de datos (normalmente relacional) y a la velocidad de interconexión de la red externa al clúster.

3.2.3. Escalabilidad.

La oferta de servicio se puede adaptar a la demanda, virtud muy deseable en sistemas distribuidos.

3.2.4. Facilidad de gestión.

Naturalmente, si el software del clúster en bueno. Se aportan elementos de planificación de trabajos, detección y prevención de averías, análisis de rendimientos, etc.

3. 3. Prestaciones a tener en cuenta en la elección de un clúster.

- Obviamente, en primer lugar la fiabilidad de trabajo.
- Velocidad de la conexión de la red que une los ordenadores del clúster.
- Posibilidad de incorporar nuevos nodos sin tener que parar el funcionamiento de todo el clúster. La reparación en caliente es pre-requisito sin escape.
- En el momento de la caída, la calidad y rapidez de la recuperación.
- El grado de adaptabilidad a la demanda.
- El reparto equilibrado del trabajo entre los nodos.
- La facilidad de administración, tema clave ya que la imagen del clúster ha de ser de máquina única.

3. 4. Clusters y aplicaciones distribuidas.

Una de la razones de montar aplicaciones distribuidas es conseguir dar servicio cuando cae un servidor. Si los servidores los colocamos en clusters, la tolerancia a fallos de la aplicación distribuida será muy alta y podremos ahorrar en software dedicado al análisis de consistencia que ha de garantizar la recuperación y funcionamiento del sistema en caso de caída en uno de los nodos. Incluso la aplicación puede diseñarse no distribuida colocando datos y software en el clúster.

La contrapartida es que si Vd. deja una aplicación condicionada a un clúster no la podrá instalar en entornos muy pequeños, restricción muy importante en aplicaciones distribuidas.

Que hacer, pues, ¿Clúster o software de consistencia? Una vez más, no hay soluciones mágicas. Mandan la importancia de garantizar servicio versus los costes de cada solución, la importancia o no de introducir un punto de heterogeneidad (el clúster lo es) o la calidad de las soluciones clúster de su plataforma habitual

Los clusters son fundamentales en aplicaciones Internet de filosofía WEB.

De cualquier forma, un buen diseñador debe conocer y valorar siempre la existencia de los clusters.

4. SAN (Storage Area Networks).

Según Garther Group, las SAN suponen la gestión centralizada de recursos y datos en un dominio de almacenamiento que proporciona servicios compartidos a un grupo de servidores y a sus aplicaciones.

Los dispositivos NAS (Network Attached Storage) se utilizan para compartir información en las redes LAN mientras que las soluciones SAN (Storage Area Networks) facilitan copias y recuperar información para optimizar el rendimiento del negocio.

SAN puede definirse como una red de fibra óptica y dispositivos de conmutación dedicados a interconectar recursos de almacenamiento y servidores sin interferencia de la red de trabajo.

Proporcionan:

- ❍ Carencia de puntos únicos de fallos.
- ❍ Alto rendimiento ya que la conexión no es por la LAN convencional con ancho de bando limitado y compartido.
- ❍ Escalabilidad, es muy fácil añadir nuevos elementos de almacenamiento.
- ❍ Disponibilidad.

Permite que varios servidores puedan acceder al mismo almacenamiento que no está conectado directamente a los clientes de la red.

Los dispositivos de almacenamiento pueden comunicarse entre si de forma que se pueden establecer sistemas automáticos de back-up sobre disco y cinta y de disponibilidad en todo momento.

El usuario o los programas trabajan con discos lógicos que el sistema SAN se encarga de maperar en dispositivos físicos de forma absolutamente transparente. El usuario ignora con que disco físico trabaja que incluso puede variar a lo largo del ciclo de vida la situación física. La unidad lógica puede contener más de un recurso físico y compartirlo con otras unidades lógicas- La flexibilidad es total.

Los dispositivos de back-up o mirroring se intercalan también de forma automática

5. Aceleradores de comunicaciones.

Son productos que optimizan el tráfico por una vía de comunicación.

Por ejemplo, si dispone de una conexión punto a punto entre dos edificios de su empresa, si instala un acelerador mejorará sustancialmente el ancho de banda útil.

Puede ser útil para la conexión remota de usuarios nómadas VIP.

Un producto muy conocido dentro de esta gama es CITRIX.

6. Netware Computing (NC).

Para enfrentarse a Windows y a Intel (dintel, en la jerga del momento), un consorcio formado por:

- **Netscape**, creada en 1995 y que pretende que el sistema operativo sea la plataforma Internet. Propone acabar con la compra de software: Internet proporcionaría todo el necesario.
- **Oracle**, que pondría los servidores de datos y los programas.
- **Sun**, que aportaría las máquinas servidoras y la plataforma de conectividad.

Propusieron una nueva forma de ver la informática desde el puesto usuario. Si:

- Primero fue el Mainframe al que los usuarios accedían mediante pantallas no inteligentes y transacciones.
- Después fue el PC, con el que el puesto cliente consiguió un lugar de trabajo autónomo.
- Más tarde la integración de los PC's en redes.
- NC propone utilizar un pequeño y barato terminal para acceder vía Internet al software cuando y donde se necesite.

La propuesta predica que desaparecen de golpe dos problemas: el coste de la compra de hardware (léase Intel) y el de las licencias del software (léase Microsoft).

La idea liga con el lenguaje actual de algunas grandes marcas, como IBM, centrado en que lo importante es el servicio, no la plataforma.

NC versus PC, es, a mi juicio, un batalla que tiene poco campo común donde poder librarse.

Los NC's son más baratos y por tanto más rápidamente amortizables y muy fáciles de administrar ya que en el fondo su filosofía de trabajo en centralista: hay un servidor de datos y un sólo servidor de programas.

Desde el punto de vista de niveles lógicos y físicos en una arquitectura distribuida, no pueden formar parte de la estructura lógica.

Sin embargo, no son autónomos cosa que les imposibilita para muchas localizaciones (donde sólo se necesita un PC) y aplicaciones que necesitan una gran adaptabilidad. Sin contar el coste de comunicaciones si los servidores de datos y programas son remotos, aunque este sea un punto actualmente a la baja. Además, si caen los servidores, se acabó la posibilidad de seguir trabajando. Y esto si que es importante.

Para un diseñador es un recurso más, pero si desea hacer su aplicación:

- Centralizada, no distribuida.
- utilizar un componente de presentación desde un servidor de programas.

En la práctica, la idea no ha tenido demasiado éxito.

7. Teléfonos móviles.

Los teléfonos móviles están íntimamente ligados a nuestra forma actual de vida. Prácticamente todos tenemos uno y llamamos ya a las personas y no a sus hogares o a sus puestos de trabajo.

El teléfono móvil puede ser un elemento activo en muchos tipos de aplicaciones distribuidas debido a ese inmenso poder de penetración.

Su uso puede extenderse obviamente a cualquier aplicación aunque es más habitual su uso en aplicaciones focalizadas en la persona y en su localización:

- Informes de eventos, en el sentido informático de la palabra, a los que se ha suscrito es propietario.
- Aplicaciones focalizadas por la localización de las personas, como la consulta de los restaurantes o librerías de la zona donde se encuentra la persona.
- Etc.

Básicamente su uso será de componentes de presentación que interactuarán a través de la plataforma distribuida con todo tipo de servicios.

8. Los asistentes personales.

Los PDA (en inglés Personal Digital Assitant) son la evolución de las agendas electrónicas. La potencia actual de estos pequeños dispositivos ha dejado atrás hace tiempo el concepto de agenda y han pasado a ser algo más.

Hoy día el PDA puede conectarse a un PC y, además de sincronizar su agenda interior con las agendas de los principales programas del mercado de este tipo (por ejemplo Microsoft Outlook), permite bajar o subir cualquier otro tipo de información.

Hay muchas organizaciones en las cuales la compra de estos dispositivos se hace desde el Dpto. de Informática para evitar la dispersión de modelos y conseguir mejores precios.

Tienen integrado el acceso a Internet y la posibilidad de conexión remota.

En cualquier caso, estos pequeños y potentes dispositivos, para los que existe versión de Windows y Microsoft Office, puede integrarse como un soporte físico más de un sistema distribuido.

La gran ventaja sobre un PC portátil es el tamaño y el precio. Su gran desventaja la programación no estándar y el tamaño de la pantalla que obliga a programar un componente de presentación especial.

¿Cuál será su futuro? Cada vez los PC's portátiles son más pequeños y potentes. ¿Llegarán a tener un tamaño y un precio equivalente a los PDA? Los Tablet PC van por aquí. Sólo el tiempo lo dirá.

Actualmente estamos en la fusión del PDA y el teléfono móvil facilitando la conexión remota a Internet y al correo. Son los PCA acrónimo de Personal Communication Asintant).

No menosprecie ni olvide el uso de PDA como soporte de funciones distribuidas siempre que esté justificado.

9. La resurrección de los Mainframe.

Ya le he comentado antes que los futurólogos que sobre 1990 asimilaron a los Mainframe a dinosaurios condenados a la extinción se equivocaron de todas, todas.

La bajada de precios y el aumento de las prestaciones que se han ido produciendo en los PC's se ha trasladado también a los Mainframe. Los precios han caído drásticamente y la potencia ha subido espectacularmente. Era iluso no preverlo.

Y los Mainframe llevaban ventaja. Eran y son estables, muy robustos, fáciles de administrar, soportan aplicaciones que ya funcionan desde años con total solvencia y con las que no se gana nada (excepto problemas) reprogramando, etc.

Y por tanto también los Mainframe tienen ya soluciones que cumplen la ecuación mágica:

Coste mínimo + potencia máxima + facilidad de administración = Servidor Perfecto

Nuevos aires han soplado sobre los Mainframe también en Software con la llegada de Internet. Las aplicaciones clásicas de Internet son por filosofía centralizadas. ¡Y que mejor housing que un Mainframe!

De la evolución (por seguir el símil biológico, que como ya se ha dado cuenta no es santo de mi devoción) ha salido una nueva especie de saurio más cercano al Tiranosaurio Rex que al Diplodocus. Y en su nicho informático se están comiendo una parte de los agresivos, inteligentes pero pequeños, Viceloraptores PC's.

Hagamos un acopio de razones en defensa de los Mainframes:

- **Amplía carga de trabajo. altísimo rendimiento y escalabilidad,** el principal argumento.
- **Entorno homogéneo.**
- **Rati inversión/prestaciones altísimo.**
- **Altísima disponibilidad.**
- **Seguridad**: son, obviamente, mucho más fáciles de defender.
- **Gestión centralizada.**
- **Mantener aplicaciones antiguas** que todavía son perfectamente válidas.

10. Comunicaciones.

10. 1. Ámbito y situación.

Tradicionalmente se conoce como comunicaciones en el mundo de la informática las conexiones entre ordenadores y/o redes remotas. Hasta los primeros años de los 90, este tipo de conexiones venían caracterizadas por tres factores básicos: interconexión, velocidad lenta y coste alto.

Con la llegada del fin de siglo y el inicio del XXI el panorama cambió radicalmente. Cuando conviene, la interconexión puede sustituirse por interoperatividad. Recursos como el RAS (*Remote Acces System*) ADSL y TELNET permiten conectar ordenadores y/o redes remotas con la misma

interoperatividad que los elementos locales, obviamente, a la velocidad que permita la infraestructura.

La interconexión dio un vuelco espectacular con la llegada de Internet. Se llega a cualquier punto del planeta de una misma forma. Y a costes razonables.

La velocidad de conexión aumenta continuamente, tanto en vías convencionales, por ejemplo las redes telefónicas conmutadas habituales, como en el desarrollo de redes de fibra óptica.

Aguzada por el aumento espectacular de la demanda, la oferta se multiplica en cantidad y calidad. Y todo ello lleva los costes hacia abajo y la llegada de tarifas planas objetivan los costes. Y además rápidamente.

10. 2. Qué necesita el diseño distribuido de las comunicaciones.

Olvidemos por un momento la infraestructura y hablemos de diseño. En aplicaciones distribuidas con puntos remotos, el diseñador se ve afectado por:

- Velocidad de la conexión, que le afecta, como ya se ha hablado, como un punto de heterogeneidad.
- Disponibilidad de la conexión, en lugar y tiempo.
- Necesidad de que los puestos remotos sean autónomos o no. No es lo mismo una aplicación de consulta en el puesto remoto que una de venta cara al público. La primera puede tener momentos de no servicio pero la segunda no. Y entre este blanco y negro hay una colección de grises.
- Si tiene suficiente con interconexión (filosofía de interfaces) o necesita interoperatividad (integración transparente).
- El coste de la comunicación a partir de dos factores: el coste fijo del enganche y el coste del consumo.
- Administración remota sin presencia física ya que los costes y el tiempo de los desplazamientos tienen mucha importancia en los prerrequisitos del diseño.

10. 3. Diseño y comunicaciones.

Y ahora cotejemos diseño con infraestructura.

- **Velocidad de la conexión**. En entornos concretos se pueden disponer de conexiones tan rápidas como las locales. Por ejemplo en lugares donde llegan cableados de fibra óptica. Y donde no llegue siempre queda telefonía con soluciones ADSL y todo la que la evolución de la técnica va sacando. En aplicaciones distribuidas: si crea una aplicación de gestión de tiendas, Vd. no sabe a priori cual va ha ser el plan de expansión, y por tanto de localización, de las tiendas de su compañía. Lo que le llevará a estar muy pendiente en cada momento de cual es la solución de comunicaciones que necesita en la siguiente etapa de crecimiento de su compañía.
- **Disponibilidad de la conexión**, en lugar y tiempo ha aumentado espectacularmente. La telefonía móvil junto a los PC's portátiles revolucionaron la situación. Esta doble combinación permite disponer de conexión remota donde y cuando se necesite. A velocidad lenta y costes no baratos.

- **Necesidad de autonomía de los puestos remotos**. En este sentido poco ha cambiado ya que depende de la naturaleza de la aplicación. Es una realidad que muchas de las clásicas aplicaciones de consulta C/S han sido sustituidas por aplicaciones Internet evitando así todo el problema de la replicación. Pero las aplicaciones que no permiten la caída temporal de servicio persisten.
- **Se dispone ya de interoperatividad e interconexión remota** para elegir cuando se necesite. La limitación de la interoperatibilidad remota es el coste y la velocidad. La ventaja de la interconexión es el estándar de Internet, en particular del correo electrónico, y su cobertura prácticamente global.
- El **coste de la comunicación** va a la baja con la competencia y las tarifas planas.
- La **Administración remota** sin presencia física es plenamente posible con la interoperatividad de las redes locales con las remotas. Y con los producto de apoyo disponibles. Aquí el coste de la comunicación queda minimizado totalmente por la disponibilidad del apoyo en el punto remoto y por la eliminación del coste de los viajes y del tiempo ahorrado en ellos. Y además, el tiempo y la calidad del servicio de soporte será inmensamente mejor.

La visión de la Infraestructura desde del Middleware.

1. Introducción.

Este capítulo es, en su mayoría, innecesario dentro de un curso de diseño distribuido. Es simple cultura general para aquel lector que quiera tener una idea de como se organizan los estándares de Middleware para conseguir la transparencia de la plataforma.

Si no siente esa curiosidad deténgase únicamente donde se introduce el concepto de Sistema Operativo de Red (NOS) y del Administrador del Sistema Distribuido (DSM), aunque de ambos se hablará más tarde desde la visión de los servicios que aportan a los diseñadores de aplicaciones distribuidas.

2. Arquitectura del Middleware.

Recordemos que el Middleware es la capa de software que se interpone entre el cliente y los servidores que proporcionan los servicios.

En la parte cliente encontraremos la interfície gráfica de diálogo con el usuario y las funciones del sistema operativo distribuido (DSM) del que hablaremos más adelante.

En el otro lado está la parte servidora donde se encuentra localizados los servicios (datos, objetos, OLTP, Groupware, etc.). Y naturalmente, la parte de DSM que corresponda.

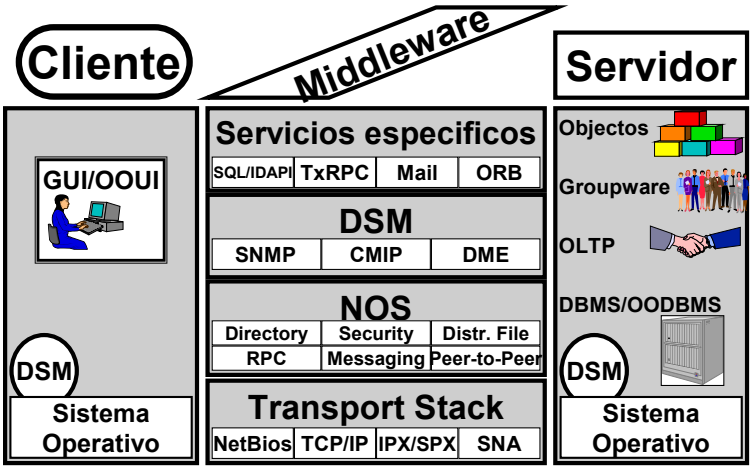


Figura 30. Infraestructura del Middleware

En medio está el Middleware, organizado en cuatro niveles:

2. 1. El nivel de Transporte (Transport Stack).

Formado por los protocolos de transporte que proporcionan comunicación *end-to-end* en las LAN's y las WAN's.

2. 2. El Sistema operativo de red (Netware Operating System).

Por encima de la capa de transporte se construye el **Sistema Operativo de Red** del Middleware (*Netware Operating System* - **NOS**).

NOS extiende las opciones locales del sistema operativo hasta incluir todos los dispositivos de red (impresoras, directorios de ficheros, módem, scanner, etc.).

NOS soporta también la comunicación entre los programas que se ejecutan sobre el sistema distribuido.

Los servicios del NOS son utilizados tanto por los clientes como por los servidores y son básicos en los desarrollos distribuidos.

2. 3. El Administrador del Sistema Distribuido (Distribuited System Administration).

Por encima del NOS se estable el **Administrador del Sistema Distribuido** (*Distribuited System Management* - **DSM**).

Su función es asumir las funciones de administración del sistema distribuido. Su importancia es notable en los diseños distribuidos ya que la administración que el DSM no resuelva habrá de aportarla con el diseño.

2. 4. Nivel de Servicios Específicos.

Aprovechando las tres plataformas anteriores se establece la capa de servicios específicos como RPC, Colas, Mail, ODBC etc.

Conviene remarcar, finalmente, que los servicios de NOS, DSM y del nivel de servicios específicos implementan el Middleware.

Observe, a nivel de curiosidad, que conjunto DSM + NOS se podía llegar por u doble camino:

- Extensión de los servicios de las redes clásicas.
- Ampliación de sistemas operativos con los sistemas operativos de red.

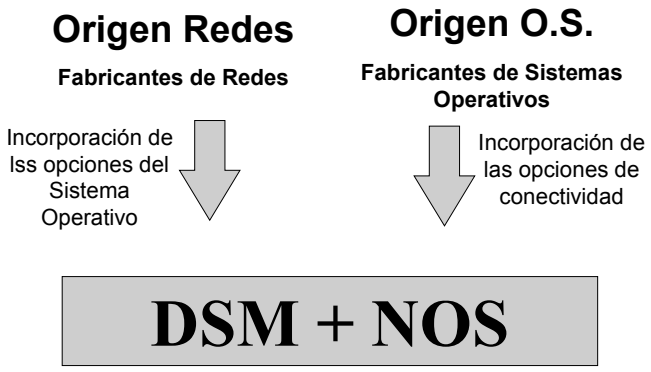


Figura 31. Origen del NOS y del DSM

Y que ha triunfado claramente la segunda vía.

Conectividad de Sistemas.

1. Concepto de Conectividad de Sistemas.

Hablar de conectividad de sistemas es hablar de las posibilidades de operarlos conjuntamente, de la interconexión entre ellos.

La conectividad es una consecuencia de las posibilidades de las redes y recursos de comunicación remota establecidos entre los sistemas ligados. Se acaba reflejando en las posibilidades que se ofrecen para trabajar los sistemas conjuntamente.

Hay dos formas básicas de mirar un sistema interconectado desde la perspectiva de las posibilidades de trabajar en él.

1. 1. Interaccesibilidad o interconexión.

Los sistemas son accesibles entre ellos pero no de forma transparente a los programas.

Si los sistemas se mantienen interaccesibles, la conexión entre ellos sólo será posible con **filosofía de interfase** y no existirán aplicaciones distribuidas si no se desarrolla Middleware específico.

1. 2. Interoperatividad.

Los sistemas son **accesibles de forma transparente** a los programas. Supone la existencia de un Middleware, comprado o construido, que lo permite. Hoy día prácticamente todos los sistemas son interoperables.

¿Qué hacer si heredamos sistemas únicamente interconectados? Si queremos hacer sobre ellos aplicaciones distribuidas hay que convertirlos en interoperables.

Hay tres caminos.

- Comprar Middleware.
- Migrar a sistemas interoperables.
- Encapsular las causas creando el Middleware en los puntos y funciones en que se necesite interoperatividad.

Prime siempre que pueda las dos primeras alternativas. Aunque a priori parezca más cara, al final ahorrará. El Middleware que compre se adaptará automáticamente a los cambios evolutivos de su SI. El Middleware que usted construya no. Vd. habrá de gastar dinero y recursos para hacerlo.

2. Conectividad y Diseño Distribuido.

El diseño de cualquier aplicación distribuida tiene como precondición que los sistemas integrados son interoperables. Si no fuera así se habrían de marcar los puntos de heterogeneidad y actuar tal como se explica al final del apartado anterior.

La conectividad de sistemas integra todo los elementos de una plataforma dentro de una plataforma interoperable.

El diseño distribuido es la forma de conseguir aplicaciones distribuidas sobre la plataforma interoperable. El diseño se apoya en las posibilidades de interoperabilidad que le proporciona la conectividad.

¡¡¡¡Conectividad y Diseño Distribuido son dos disciplinas informáticas diferentes!!!!

Si Vd. lector esperaba encontrar aquí un curso de como conectar sistemas vaya a devolver lo que está leyendo rápidamente. Espero que recupere su dinero.

El diseñador de aplicaciones distribuidas debe conocer las posibilidades de conectividad de las plataformas de las que dispone y de las que existen en el mercado. Pero no ha de caer nunca en la trampa de confundir las dos actividades.

¿Como obtener, pues, esta formación? Haga un curso sobre las posibilidades de interoperatividad de los sistemas actuales. Sin embargo el consejo no le será fácil de seguir ya que encontrará muchos cursos de administración pero pocos de funcionalidad.

Existe otra vía. Sea curioso en su experiencia del día a día. E intente aprender de esa experiencia. Asista a todos las sesiones comerciales de presentación de productos de conectividad que pueda. En ellos, debido a lo apurado del tiempo disponible se habla sólo de posibilidades y no de administración.

Sin embargo no se crea de la misa la mitad....y filtre lo que oiga con la experiencia de otros profesionales,

3. Arquitectura física y lógica.

La arquitectura física de una arquitectura distribuida viene determinada por las capas de conectividad de la plataforma del sistema.

Por ejemplo, una arquitectura física de cuatro niveles puede estar compuesta por:

- Un HOST.
- Uno o varios servidores en al central que actúan de Front-end conectado/os al HOST.
- Servidores departamentales conectados a los servidores de la central.
- Redes de PC's conectados a cada servidor departamental.

La **arquitectura lógica** está determinada por los niveles o capas donde las aplicaciones pueden localizar recursos (datos y procesos) gestionados de forma interoperable a través de Middleware. La arquitectura lógica resulta de la física

filtrándola con los puntos donde la organización permite o necesita colocar administración.

Hay diferentes criterios para establecer la arquitectura lógica sobre la física:

- La interoperatibilidad del Middleware
- La estructura empresarial.
- La dispersión geográfica.
- La política de administración del sistema distribuido.
- Necesidades específicas de cada aplicación.
- Etc.

En la figura se ha representado la arquitectura física de cuatro niveles referenciada con anterioridad.

Analicemos diferentes situaciones:

a) Se establece una arquitectura Web soportada por el servidor de la central sobre el que actúan directamente los usuarios a través de un navegador. Estamos delante de una arquitectura lógica de un nivel.

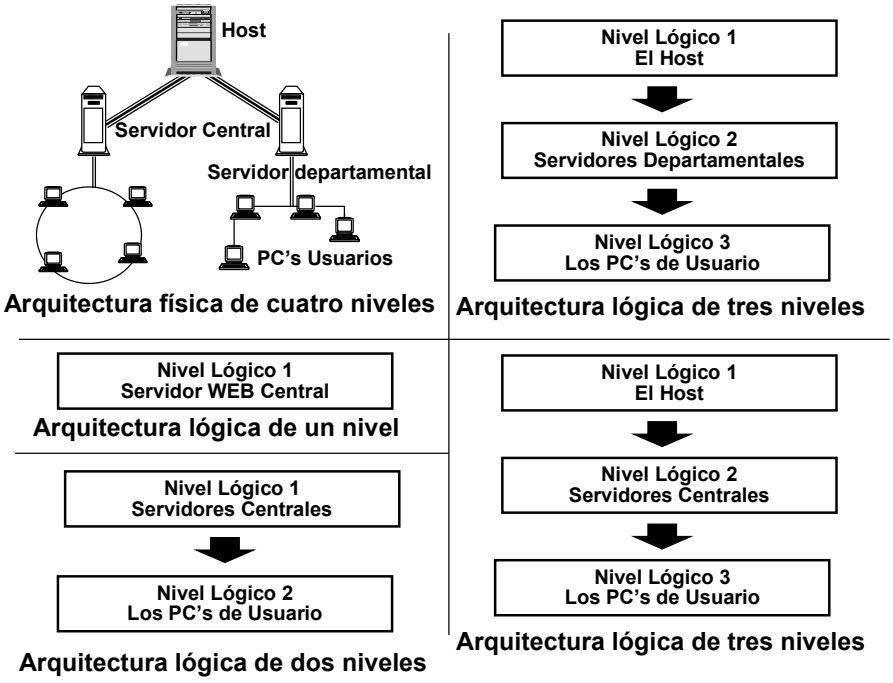


Figura 32. Relación entre las arquitecturas lógica y física de una plataforma distribuida.

b) La aplicación se localiza sobre el servidor de la Central y el resto de servicios están sobre los puestos clientes. Estamos, obviamente, ante una arquitectura de dos niveles.

c) No se quiere montar administración de sistema sobre el servidor de la central que se dedica únicamente a router de comunicaciones por lo que no se instalarán servicios sobre ese nivel físico. Se establece entonces una arquitectura lógica de tres niveles: HOST, servidores departamentales y PC's de usuario. En el nivel HOST se instalan los servicios corporativos, en el nivel departamental los específicos de cada departamento y en el nivel usuario sólo aquellos que garantizan independencia ante caídas del servidor o que aumenten el rendimiento del sistema.

d) Imaginemos ahora que hemos realizado una política de concentración de servicios, aprovechando el aumento de ancho de banda de la plataforma de comunicaciones, y hemos rescatando los servicios antes localizados en los servidores departamentales alojándolos en los servidores centrales. La función es concentrar la administración para minimizar costes. Seguimos estando delante de una arquitectura de tres niveles simétrica al caso anterior donde la función de los servidores centrales y departamentales se ha intercambiado.

Las combinaciones son tantas como posibilidades o características tenga el sistema distribuido. Por ejemplo, si no hay necesidad de dotar a los PC's de usuario de autonomía de trabajo en la aplicación cuando no funcionen los servidores departamentales, la arquitectura lógica puede reducirse únicamente a dos niveles: HOST y servidores departamentales.

Cada aplicación puede tener un número de niveles lógicos diferente. Se tomará como **profundidad de la arquitectura lógica** el máximo de niveles de cada una de las aplicaciones que funcionan sobre la plataforma distribuida. Es muy recomendable definir la profundidad del sistema distribuido con independencia de las aplicaciones que funcionan.

Obviamente existe una relación entre la arquitectura física y la arquitectura lógica, pero si la arquitectura distribuida está bien pensada y construida las dos arquitecturas son independientes, únicamente relacionadas por la localización de los servicios. Recordemos que esa independencia la garantiza el Middleware.

Cada compañía habría de establecer un **Plan Estratégico de su Arquitectura Distribuida** con las arquitecturas física y lógica que soporta. A partir de él puede planificarse el modelo de crecimiento de la infraestructura.

La arquitectura lógica marcarán los niveles que los diseñadores de aplicaciones deberán tomar como precondiciones para distribuir y localizar los servicios de datos y procesos. Esta precondición se tomará, como se verá más adelante, como precondición del diseño de la arquitectura distribuida del sistema que se tratará más adelante.

4. Aspectos de la conectividad que afectan al diseño.

- Niveles lógicos de la arquitectura que permiten la conectividad. Recuerde que allí es donde podrá localizar servicios, datos o procesos.
- Introducción de puntos de heterogeneidad. Por ejemplo una comunicación remota lenta.
- Buscar elementos de conectividad que puedan ayudar. Por ejemplo sustituir una conexión lenta (Gateway) entre dos servidores por otra rápida (un Router), utilizar Clústeres, localizar servidores en una máquina UNIX, etc...
- Clústeres y Mainframe como proveedores de servicios fiables y potentes.
- La topografía de la red
- Las posibilidades de los dispositivos móviles.

5. Procesos Distribuidos y procesos Distribuibles.

La terminología de proceso distribuible y proceso distribuido es un clásico.

Se conoce como **proceso distribuido** aquel en que la funcionalidad y/o los datos que lo conforman pueden residir de forma transparente en cualquier punto de la plataforma de la arquitectura distribuida.

Se conoce como **proceso distribuible** aquel en el cual, si bien las funciones y/o los datos pueden estar distribuidos por la plataforma, la distribución no es transparente y está ligado a puntos concretos de la plataforma.

Así, un proceso que utilice directamente el gestor de BD de un AS/400 puede situar ese gestor en cualquier AS/400 de la red, pero sólo en AS/400. Será sólo distribuible. Un proceso que ataque un gestor SQL vía ODBC podrá atacar de forma transparente cualquier motor de BD que admita ese interfície, hoy día todas. El proceso se dirá distribuido.

Es evidente que los procesos distribuidos son preferidos a los distribuibles. Sólo el rendimiento (perfomance) o la existencia de un sistema heredado puede recomendar hacer un proceso distribuible y no distribuido.

El diseño distribuido está en deuda con la terminología distribuida y distribuible ya que su sola enumeración supuso el inicio del camino que llevó al concepto de Middleware. Rindo aquí tributo a esa deuda.

6. Fundamentos físicos de conectividad.

A modo de resumen de final de esta parte, observe que las posibilidades de la conectividad están ligadas a tres elementos:

6. 1. La infraestructura.

O concreción física del sistema, de la que ya se ha hablado.

6. 2. Las redes y las comunicaciones.

Por descontado, siempre desde la perspectiva diseño.

6. 3. Los protocolos y los productos de Middleware.

De hecho, los programas deben trabajar con el Middleware y no con los protocolos. Más adelante se hablará también de este tema.

Cliente/Servidor, Internet y Proceso Centralizado

1. Introducción.

Primero fueron las aplicaciones centralizadas HOST, transaccionales y por lotes mediante cadenas Batch.

Después las aplicaciones departamentales que, interconectadas por interfaces con el ordenador principal, tenían fundamentalmente el mismo diseño centralizado.

Mas tarde aparecieron las aplicaciones distribuidas Cliente/Servidor basadas en sistemas operativos, originalmente conocida simplemente como aplicaciones C/S.

Unos las tildaron de moda pasajera. Otros pronosticaron solemnemente la muerte del proceso HOST. La realidad, y los buenos informáticos, pusieron cada cosa en su sitio. HOST y C/S convivieron, colaboraron y se reforzaron mutuamente permitiendo a los informáticos dar mejores soluciones a sus diseños. Naturalmente, existía una zona gris entre los territorios naturales de ambos modelos que permitió seguir publicando artículos y libros a favor y en contra.

Y llegó Internet. Y después del simple acceso a WEB, las aplicaciones activas sobre Internet. Y la historia se repitió. Unos las tildaron de moda pasajera. Otros pronosticaron solemnemente la muerte del C/S convencional. Los "centralistas" confirmaron eufóricos que tenían razón: C/S clásico sólo había sido un engorro pasajero. Y sin observar que Internet es una arquitectura C/S por si misma.

Los "centralistas" afirmaron que como los programas de las aplicaciones Internet están almacenados en un servidor centralizado y presentan datos de esa misma naturaleza, son, básicamente, ¡de diseño y administración centralizados! De presentación gráfica, pero diseño centralizado. ¡El HOST ha vuelto! El círculo parece haberse cerrado. O el pez se ha mordido la cola, como prefiera.

Calma. Creo que es el momento ideal en nuestro viaje para poner cada cosa en su sitio.

Y aprovecharé este capítulo para introducir el concepto de **servicios pasivos**, servicios **activos** y el de **agente**.

2. ¿Qué es Internet?

Igual que cuando hablamos en un capítulo anterior de redes, no le voy a hacer una presentación exhaustiva de Internet. Vd. lector conoce, seguro, que es Internet.

Solo recordar, para empezar a hablar, que Internet es una red que enlaza todas las máquinas que lo deseen. Es también conocido que Internet ya existía como tal desde hacia mucho tiempo aunque su popularización fue posterior. No voy a escribir el número gigantesco de nodos de la red que aparece siempre en párrafos de este tipo para dar importancia a la red (siempre me ha intrigado que método se ha utilizado para contar y justificar ese número).

La importancia de Internet no está en el número de sus nodos sino en la generalidad y estandarización de la conexión.

Todo el mundo que lo desee se puede conectar. Y desde cualquier punto de la Tierra donde llegue alguna vía de telefonía, aunque como ya sabe hay otras vías. Porque si no hay infraestructura telefónica, física o móvil, cosa prácticamente imposible hoy día, siempre quedan los satélites.

El nexo común de la red es el protocolo TCP/IP que resuelve la heterogeneidad de cada sistema individual.

Sobre esta plataforma de transporte se han construido los protocolos que dan los servicios de la red:

- SMTP para el correo electrónico.
- TELNET para la conexión de máquinas remotas. O locales si conviene.
- FTP para el intercambio de ficheros.
- HMTL como formato estándar de presentación.
- CGI, en claro desuso, y Servicios WEB como petición de servicios.
- XML que se ha convertido en un estándar de intercambio de información entre procesos.
- SOAP para llamadas remotas RPC con descripción del servicio en XML.
- Y un largo etcétera en evolución continúa.

3. ¿Qué es una WEB?

Aunque la palabra que triunfó fue la de WEB, inicialmente se usaba WWW iniciales de *World Wide Web*. Se hacía difícil pronunciar tres W's seguidas (letra que ni siquiera existe en castellano) con lo que el término WEB se impuso.

El concepto de *World Wide Web*, que puede traducirse por algo así como telaraña universal, fue creado en 1990 en Suiza por Tim Berners-Lee, en aquel momento un joven estudiante del Laboratorio de Física de Partículas (CERN).

Una WEB (usaré esta palabra a partir de ahora) es un sistema de distribución y presentación de información bajo Internet en páginas de hipertexto. La información se puede publicar instantáneamente. Y navegando por la red se puede acceder desde cualquier nodo conectado a cualquier servidor conectado.

Hoy día la WEB es un verdadero sistema multimedia con gran capacidad de atender clientes muchos clientes a la vez. El acceso simultáneo de muchos clientes al mismo servidor se conoce como concurrencia. Internet tiene unos índices de concurrencia enormes y proporciona los recursos para soportarlo.

4. Las aplicaciones Internet pasivas.

Con la plataforma Internet generalizada y operativa, utilizarla para hacer proceso de aplicaciones era inevitable.

Surgieron así aplicaciones Internet caracterizadas por:

- Existencia de programas residentes en el servidor de la WEB.
- Accesos por los clientes desde un navegador utilizando una URL.
- Recepción del componente de presentación de esos programas en el nodo de acceso cuando se realiza la conexión a través del navegador.
- Manipulación del componente de presentación por el usuario y envió al servidor Web para su proceso.
- Los datos son procesados en el servidor Web y el componente de presentación es enviado actualizado al cliente, normalmente con los resultados de una consulta, a aceptación o los errores de los datos registrados o una ampliación de la petición de datos.
- Gestión desde estos programas de aplicaciones de:
  - Consulta de BD o de almacenes de datos (periódicos, revistas, normativas y un etc tan largo como desee) residentes en la zona de influencia de la WEB.
  - Captura de datos de un usuario del nodo conectado y registro en la zona de influencia de la WEB.

Son ejemplos habituales del primer modelo, llamado de **consulta,** aplicaciones para poner datos estadísticos al acceso de usuarios y del segundo modelo, llamado de **captura de datos,** aplicaciones de registro de pedidos sobre verdaderas tiendas virtuales.

Observe que este modelo de aplicaciones Internet, que voy a denominar **modelo de aplicaciones Internet Pasivo**, tiene una característica básica: no hay ninguna interacción con los datos y programas del nodo conectado. El navegador utilizado en el nodo de conexión es un mero recurso de trabajo.

El modelo pasivo tapa, de entrada, un hueco no cubierto hasta ese momento: la ejecución de aplicaciones sobre plataformas desconocidas al sistema informático desde donde se ofrece la aplicación. **Solo eso ya fue un avance fundamental**.

Existe otro avance no valorado, a mi juicio, como se merece: la estandarización de la interfície gráfica. Hablaré de ello más adelante.

Evidentemente, este modelo es una alternativa válida para las aplicaciones C/S convencionales de consulta, basadas en sistemas operativos, ya que evitan la siempre problemática replicación de datos de la que habremos de hablar exhaustivamente más adelante.

La información está disponible inmediatamente y sin problemas de inconsistencias. Además, en este tipo de aplicaciones, la disponibilidad y la estandarización está ligado a la rapidez de acceso. Y permiten igual formato si se trabaja internamente en la compañía y si se deja acceso externo. El único problema es el posible coste de las comunicaciones. Pero en la mayoría de las ocasiones, el coste queda rápidamente amortizado por la eliminación de los costes de administración y la aparición masiva de las ofertas de tarifas planas.

Para este tipo de aplicaciones de consulta el diseñador podrá elegir entre el modelo distribuido basado en Internet o Sistemas Operativos en función de la aplicación concreta. Por ejemplo, si los datos de consulta son históricos y su acceso es muy frecuente y variado, potenciará la replicación siempre que la administración no condicione costes altos de explotación. Si la información es volátil y los usuarios son estables probablemente será mejor Internet. Y en medio quedará la inevitable zona gris que habrá que decidir aplicación por aplicación según las características,

la arquitectura informática y la organización de cada compañía. Cuando los usuarios son nómadas se deberá valorar que será mejor.

Las aplicaciones distribuidas basadas en Internet de consulta son también una alternativa eficaz para las consultas transaccionales sobre Mainframe. Sobre todo desde que los resucitados Mainframe son servidores WEB perfectos. ¡Qué mejor que consultar los datos desde la WEB donde han estado siempre!

Las aplicaciones de captura de datos se han convertido en el recurso fundamental de muchísimas empresas comerciales. Aquí las aplicaciones distribuidas basadas en Sistema Operativo (aunque el modelo de comunicación sea Internet) cuando las plataformas son propias o, como mínimo, hay control sobre ellas, tienen ventajas sobre las aplicaciones Internet. En contrapartida, si la aplicación debe ser accesible por clientes desde máquinas desconocidas, Internet será mucho más adecuada. Si los dos casos se dan simultáneamente, Intranet e Internet son el camino a seguir.

5. Aplicaciones Internet Activas.

La siguiente evolución de las aplicaciones Internet clásicas fue también inevitable.

Ya que el programa se ejecuta sobre una máquina remota, ¿por qué no interactuar con los datos y los programas de esa máquina?

Son las aplicaciones que denominaré **modelo Internet activo** donde las aplicaciones WEB interaccionan con el entornote datos y procesos locales.

No sé si soy capaz de transmitir al lector la importancia y trascendencia de este hecho. Por un lado el proceso centralizado de las aplicaciones Internet clásicas pasa a ser distribuido. Y por la otra... ¡PELIGRO! ¡Un programa desconocido externo tiene acceso a tu ordenador!

La segunda característica es un problema a resolver por los técnicos de sistemas y los fabricantes de antivirus y cortafuegos.

El primero tema si que entra dentro de mi especialidad: los modelos de aplicaciones distribuidas.

De hecho, las aplicaciones distribuidas hoy día han de verse como un diseño unificado con dos formas de implementación de servicios Cliente/Servidor, los basados en Sistema Operativo y Internet.

A lo largo de este libro iremos comentados las cosas comunes y las características diferenciales de cada una de las dos formas de implementación.

6. La estandarización de la interfície gráfica de usuario.

Hay un aspecto marginal pero fundamental en las aplicaciones Internet: una propuesta “de facto” para la estandarización de la interfície gráfica de usuario.

Hasta ahora existía la omnipresencia de la GUI popularizada por Windows. Pero si Vd. quería construir una aplicación con interfície GUI que además de Windows

corriera en UNIX, AS/400 o cualquier otra máquina tenía un problema grave. El estándar de Internet le ha resuelto, **¡al fin!**, el problema.

Con Internet este problema ya no existe; ejecutando el programa sobre el navegador, su programa queda disponible sobre cualquier plataforma.

7. La doble visión de Internet para las aplicaciones distribuidas.

Como es posible que después de leer el apartado anterior tenga Vd. dudas sobre las relaciones C/S e Internet, y como dejar desde aquí este tema claro es clave, le propongo la siguiente reflexión.

Es decir, Internet tiene un doble uso:

- Una forma alternativa y complementaria al Cliente/Servidor basado en Sistema Operativo para diseñar aplicaciones distribuidas:
  - Como aplicaciones Web pasivas.
  - Como aplicaciones C/S basadas en Internet (activas).
- Vía de comunicación de aplicaciones distribuidas Cliente/Servidor convencionales basadas en Sistemas Operativos.

Para seguir este camino conviene que repasemos, siempre desde el punto de vista del diseñador, el mecanismo de transferencia que aporta Internet.

8. Mecanismo de transferencia en el modelo pasivo.

Repasemos como se produce el mecanismo de transferencia en el ámbito de una WEB. Se muestra un esquema en la figura.

1. Una vez levantada una vía de comunicación, se arranca un navegador en el nodo del Cliente WEB.
2. El usuario especifica desde el Cliente WEB y a través del navegador la dirección de la página WEB que quiere consultar.
3. El cliente establece la conexión con el servidor WEB que le inicia o no una sesión y en este caso le asigna una pagina de datos
4. El cliente, tras autentificarse si es necesario, solicita la página o objeto deseado recibiendo un componente de presentación.

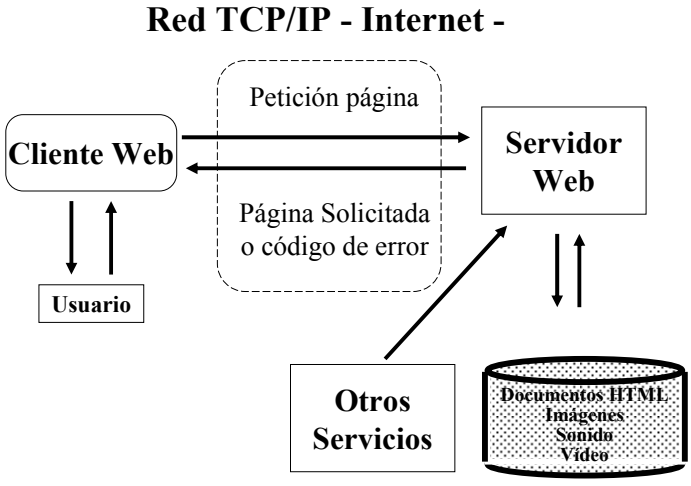


Figura 33. Mecanismo de transferencia Internet WEB.

5. Cuando el cliente envía la página cumplimentada al servidor WEB, el servidor le asigna una línea de proceso para atenderle, recuperando, si ha lugar su página de datos. A partir de aquí, localiza la información o el proceso pedido y los devuelve al cliente. En caso de no localizarlos le envía un código de error.

La página de datos, si la hay, se actualiza y se guarda para el siguiente acceso.
6. El cliente interpreta y presenta la información al usuario.
7. Se rompe la conexión.

El **comportamiento** del servidor WEB es, como queda muy claro, descaradamente **transaccional**.

Dentro de los mecanismos de conexión existen otros componentes de interés:

- Enlaces hipertexto, que permiten navegar e integrar información de diversos servidores.
- Servicios WEB del que hablaremos a continuación.
- API's especificas, etc.

Remarquemos la importancia del multimedia, del diseño artístico y del marketing dentro del diseño de una WEB.

Sin embargo**, lo que hace que una WEB sea visitada habitualmente no es la final la forma sino el contenido y su permanente actualización**. La forma, importante, es de segundo nivel.

9. Componente de Integración.

Los componentes de integración permiten el intercambio de datos y servicios entre el cliente y los servidores a través de la WEB. Son, pues, una forma de comunicación C/S.

La arquitectura de diseño se base en la existencia en el servidor WEB de unos componentes capaces de recibir peticiones del cliente, normalmente datos y petición de servicios, acceder a los recursos del entorno donde está el servidor WEB, y desde aquí a donde haya que llegar, y devolver la respuesta al cliente.

El mecanismo de transferencia se muestra en la figura. Consiste básicamente en:

1. El usuario entra la petición sobre el cliente WEB.
2. La plataforma Internet lleva la petición hasta el servidor WEB.
3. El servidor WEB lleva la petición el componente de integración.
4. El componente de integración resuelve la petición y prepara la respuesta en una página HTML.
5. La plataforma Internet lleva la respuesta hasta el cliente WEB.
6. El navegador muestra al cliente la página HTML.

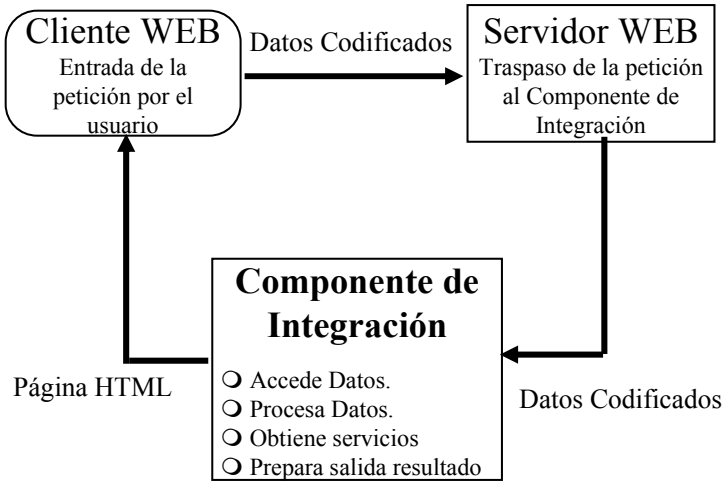


Figura 34. Comunicación entre un cliente y un Servicio a través de la WEB

Observe que está utilizándose una arquitectura C/S con la que el componente de integración accede a los servicios que necesita para resolver la petición.

10. Servicios WEB.

Un Servicio WEB es un mecanismo que permite es acceso a servicios de forma transparente a la través de la red

Ni más ni menos, y una vez más, que un verdadero mecanismo cliente/servidor implementado a través de Internet

Este modelo de petición de servicios es lo suficientemente importante como para que le dediquemos una atención especial en el capítulo siguiente.

11. Internet como una vía para el diseño C/S.

Arquitecturas como las que hemos presentado permiten que la aplicación pueda hacer varias tipos de proceso:

- Cliente suministrado por la WEB, o **Cliente WEB**
  - **Directo**. El componente de integración procesa directamente la petición interactuando con la WEB a través de Servicios WEB, saltando el componente de presentación. Observe que si no es a costa de hacer del componente una "gran aplicación" poca cosa puede hacerse.
  - **Delegación.** El componente de integración aporta lógica de negocio y obtiene por cliente/servidor los servicios, de proceso y datos, necesarios para completarla**.** En la segunda parte veremos que hay otras formas de arquitectura de servicios además de la delegación y que pueden ser utilizadas aquí
  - **Pasarela**.
    - El componente de integración se limita a recibir y validar la petición a que la WEB la traslade a otros programas, normalmente servicios, que son las que la procesan.
    - Estas aplicaciones devuelven la respuesta al componente de integración que es la que devuelve la aplicación al cliente WEB. Es el caso habitual.
    - La lógica de negocio es mínima.
- El componente de integración es en realidad un servidor que llama a los componentes locales. Es la arquitectura de **Servidor WEB**. La diferencia entre Servidor WEB y Cliente radica en la presencia o no de lógica de negocio. El servidor WEB se establece como enlace con otros componentes internos que proporcionaran el servicio y se limita a preparar la respuesta para el navegador. No puede utilizarse directamente desde el navegador.
- La **Web solo suministra el servicio**, normalmente como un Servicio WEB. El proveedor del Servicio WEB desconoce quien, como y donde se ha creado en cliente final. Se diferencia de los dos casos anteriores en que en la parte cliente puede haber un programa convencional, basado en sistema operativo, que obtiene el servicio a través de Internet.

Los clientes WEB tienen dos partes, el componente de presentación que se envía al puesto cliente y se ejecuta bajo control del navegador y el componente que queda en el housing de la WEB.

Observe ahora la figura de la derecha que resume la idea anterior. La arquitectura mostrada no es más que una de las varias posibilidades. O el servicio es claro a través del Servicio WEB o el trabajo del componente de integración obtener las respuestas o distribuir las peticiones entre los diferentes servidores de los servicios. Estamos delante de una autentica y verdadera aplicación C/S.

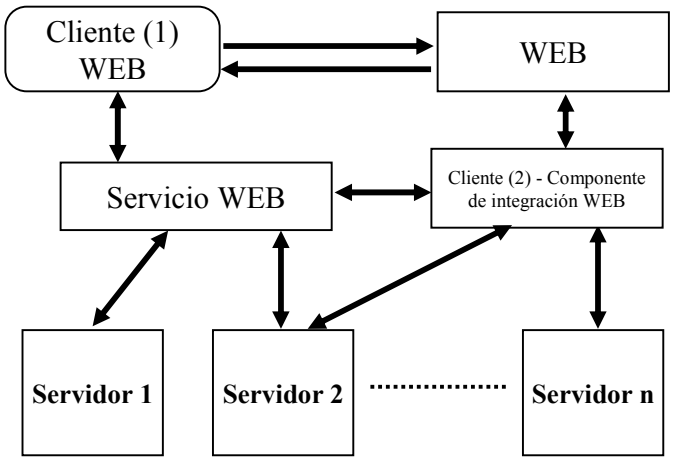


Figura 35. Internet como transportista C/S.

Internet y C/S se han apoyando y reforzado mutuamente.

Observe que Internet actúa también de transportista. No le he introducido todavía el concepto de arquitectura de servidores. Pero recuerde, para cuando lo haga, que en esta arquitectura el componente de integración, el cliente y el servidor Web establecerán entre ellos un modelo de comportamiento, una arquitectura de integración. De este tema hablaremos intensamente en la segunda parte.

12. Prestaciones de los Clientes WEB.

Además, la arquitectura basada en un Cliente WEB proporciona prestaciones básicas que según las necesidades de la aplicación distribuida pueden ser muy útiles.

- El cliente WEB actúa como cliente de presentación de la aplicación distribuida con la gran ventaja que se ejecuta en cualquier ordenador con navegador aun desconociendo la plataforma cliente. Además la operativa de una GUI Internet es un estándar que todo usuario sabe usar.
- El cliente WEB es el mismo para clientes locales que remotos.
- No hay problemas de mantenimiento de versiones ya que el servidor WEB hace de servidor de programas. Sin embargo, si Internet cae por la causa que sea, el cliente se queda sin servicio. Como ya sabemos hay aplicaciones distribuidas en las que esto no es admisible.
- El alcance de la aplicación distribuida es tan amplio como la red. Y no hay que hacer ninguna instalación en la máquina cliente para que la aplicación funcione. Basta en Navegador.
- La aplicación distribuida se ha integrado con el resto de la instalación. Si las anteriores aplicaciones distribuidas ya disponían de los servidores creados, Internet los utiliza rápidamente. Si no es así, los servidores construidos quedan a disposición de todas las aplicaciones. **Internet y C/S se han integrado perfectamente**.

13. Servidores WEB.

Como he dicho, existe la posibilidad de otra arquitectura mucho menos frecuente pero también viable. Es la que se muestra en la figura siguiente.

En esta arquitectura, el cliente WEB se ha transformado en un Servidor WEB Cliente que explora una cola en la parte cliente (componente que, aunque todavía no he presentado es lo que Vd. ya intuye, una lista de espera de peticiones pendientes) y enlaza con el servidor WEB de distribución. En esta arquitectura de servidores el Servidor Cliente WEB hará una "delegación de servicio" al servidor WEB de distribuidor residente en la WEB.

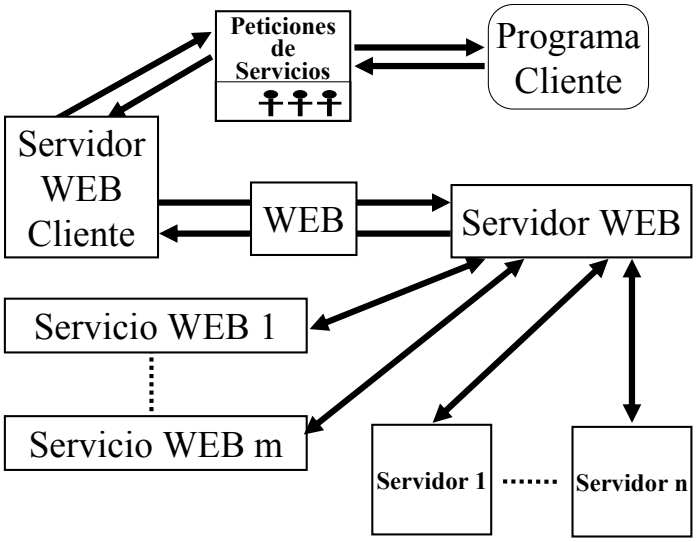


Figura 36. Servidor Cliente WEB

Una de las ventajas operativas de esta arquitectura es que la WEB puede "enviar el servidor" a la parte cliente que puede funcionar sin conocer la plataforma a donde ha llegado y controlando perfectamente su versión.

Nótese que con esta arquitectura tecnológica, **el cliente dispone siempre de la última versión del servicio, tanto en depuración de errores como en nuevas prestaciones.**

Por último, no confunda esta arquitectura con un Servicio WEB en la cual le servicio se pide directamente a la red.

14. Internet y el proceso centralizado.

Con lo expuesto hasta ahora en este capítulo, la relación entre Internet y proceso centralizado es fácil de abordar.

Las aplicaciones Internet pasivas cooperan y refuerzan positivamente a las aplicaciones centralizadas en HOST.

Las empresas de filosofía de trabajo centralizada, que se habían visto obligadas a delegar en C/S sus aplicaciones de consulta y captura de datos sin mucho convencimiento y a cambio de aumentar sus costes de administración, han visto en las aplicaciones Internet la solución a sus problemas reinstalando sus aplicaciones en HOST y dejándolas disponibles por Internet para sus usuarios remotos vía Clientes WEB.

Por otra parte, las empresas que habían mantenido sus aplicaciones de forma numantina en el HOST han visto, al fin, una salida a sus necesidades con la utilización de aplicaciones Internet.

En ambos casos, las interfícies transaccionales de las aplicaciones HOST tradicionales pueden sustituirse con componentes de presentación gráficos, en un proceso de reingeniería que denominaremos maquillaje.

Existen también productos de emulación de terminal a través de WEB que permiten el uso de aplicaciones HOST transaccionales sin necesidades de disponer y configurar un emulador en el PC del usuario.

El resurgimiento del Mainframe como servidor WEB con su extraordinario ratio potencia/coste ha sido fundamental en la extensión de las aplicaciones WEB en el mundo del HOST. Y la necesidad de disponer de servidores WEB de gran potencia para apoyar estas decisiones ha ayudado al resurgimiento del Mainframe. Una vez más la sinergia mutua entre las dos filosofías, centralizada e y distribuida basada en Internet, se potencian mutuamente.

Finalmente, los Servicios WEB, de los que hablaremos a continuación, son una forma de poder proporcionar al exterior servicios de aplicaciones Mainframe.

15. Los dispositivos móviles.

El uso de los dispositivos móviles abarca dos grandes áreas.

15. 1. Inalámbrica.

Responde a la idea que el usuario de esos terminales se mueve en el entorno de red local y que el uso de un Terminal inalámbrico corresponde solo a una necesidad de la aplicación.

En este entorno hay básicamente dos bloques de recursos:

- PC's portátiles, que a efectos de diseño se comportan como un PC más.
- Asistentes personales, más pequeños y manejables, que instancian componentes de presentación o clientes muy ligeros ya que están en contacto permanente con sus servidores.

Recuerde que con conexión inalámbrica obtendrá velocidades de transmisión en la red mucho más pequeñas lo que puede convertirse en un punto de heterogeneidad.

15. 2. Nómada.

Hay dos tipos básicos de aplicaciones que no tienen conexión permanente y sin limitaciones con el sistema distribuido:

- Aplicaciones convencionales en los que se usan PC's portátiles. Por ejemplo, un vendedor ambulante que realiza una ruta y se conecta periódicamente con la central.
- Aplicaciones de alta movilidad, donde las necesidades del sistema distribuido en el puesto nómada son muy concretas y se resuelven a través de una aplicación especializada. Por ejemplo el seguimiento de la ruta de un camión de reparto. Corresponden habitualmente a asistentes personales o teléfonos.

En el entorno de aplicaciones nómadas las hay de dos tipos:

- **Completos**: hay periodos en que están desconectados del sistema distribuido.

- **Limitados**: tienen conexión permanente pero el ancho de banda es limitado, por razones físicas o operacionales. Obliga al diseñador a introducir en sus diseños el mínimo de accesos al sistema distribuido, probablemente ayudándose en la replicación de la información no volátil.

Por sus características, asistentes personales y teléfonos se especializan en aplicaciones diferentes.

15.2.1. Asistentes personales.

Son básicamente aplicaciones donde el asistente sustituye a un PC por su menor tamaño, coste y facilidad y rapidez de ampliación.

Pueden incluirse aquí aplicaciones del tipo:

- Aplicaciones de consulta a través de WEB.
- Aplicaciones de captura de datos. Suponen habitualmente una conexión, una carga, una desconexión, la captura de los datos y una posterior descarga de lo recogido con otra conexión y desconexión. Un ejemplo puede ser la lectura de consumo de la luz, donde se vuelca la ruta y el consumo anterior, se lee con terminal en mano, y se vuelva lo leído. No hace falta decir que la conexión puede local o remota.
- Aplicaciones en que el asistente instancia el componente de presentación de una aplicación de mantenimiento de datos on-line. Esta posibilidad está muy limitada por el tamaño de la pantalla.

15.2.2. Telefonía.

Son aplicaciones basadas en la idea de que todo el mundo tiene un móvil o puede disponer de uno fácilmente.

Pueden incluirse en este grupo aplicaciones del tipo.

- Basadas en la localización de personas o objetos. Por ejemplo, aplicaciones para conocer la situación de una flota de camiones o de reparadores de averías. En el primer caso, los camioneros pueden recibir instrucciones sobre el tiempo o el tráfico y en el segundo de cual es la siguiente visita que hay que realizar. También se pueden incluir aquí aplicaciones que avisen de ofertas en los centros comerciales.
- Acceso a iniciativa del usuario a información de la zona donde se encuentra: restaurantes, oficinas de correos, etc.
- Información urgente de recibir, como la modificación de una cotización en bolsa o la disponibilidad de una información que puede afectar al negocio.
- Suscripción a eventos como resultados deportivos, noticias, etc.
- Servicios de marketing y fidelización.
- Servicios de compra muy específicos y pago a través del teléfono.

16. Servicios activos y pasivos: rutinas, servidores y agentes.

Existe otra visión de los servicios, complementaria de todas las demás, en función de su actitud en el sistema distribuido.

Un servicio se dirá que es **pasivo** si sólo actúa si es llamado por otro componente.

Es el servicio de toda la vida: conseguir el estado del crédito de un cliente, encriptar, eliminar un cliente de la BD y un largo etcétera tan extenso como quiera.

Observe que un servicio pasivo, ese servicio sólo actuará si es llamado por otro componente.

Existen dos formas de implementar servicios pasivos:

- **Estáticamente**: **linkarlos** en el programa: las **rutinas o subprogramas** de toda la vida.
- **Dinámicamente: llamarlos en tiempo de ejecución con un protocolo C/S. El servicio lo implementa un servidor.**

Muchas veces, el paso de estático a dinámico es únicamente encapsular en servicio estático en un servidor incorporándole un mecanismo de llamada dinámica.

Por el contrario, un servicio se dirá activo si toma la iniciativa en función de eventos del sistema.

Por ejemplo, un servicio que facture en función de pedidos y albaranes obtenidos desde un proceso Internet donde los inicie el propio cliente, una entrada manual por el personal de servicios centrales o un proceso batch. El servicio estará activado en el sistema esperando el evento que le informe que hay trabajo por hacer y lo realizará de forma desacoplada con los procesos del entorno distribuido que le han generado el trabajo.

La implementación de servicios activos se hace por **agentes**. Los agentes serán los encargados de ejecutar servicios desacoplados, es decir servicios que se piden y de los que no se espera respuesta: el agente los ejecutará cuando corresponda y el programa que los ha pedido no se esperará.

Observe que el programa que los pide necesita seguridad absoluta de que se ejecutaran sin problemas. Más adelante hablaremos en profundidad de todo ello.

Por último, recuerde también el concepto de servidor como máquina donde se localizan servicios.

Generaciones de Aplicaciones Distribuidas

1. Introducción.

Es un buen momento para repasar que tipo de aplicaciones se pueden montar distribuidas según los modelos C/S basados en Sistema Operativo e Internet. Y empezar a entender el por qué.

La solución tecnológica de estas aplicaciones ha evolucionado siguiendo la tecnología informática sobre las que se han desarrollado. Primero C/S basado en Sistema Operativo y, más tarde, también basado en Internet. Pero no la tipología de las aplicaciones distribuidas que con ellos se montan que sólo se han ampliado

La evolución de las diferentes tecnologías, adaptándose a la evolución informática y las nuevas propuestas informáticas, para mejorar las facilidades y prestaciones de distribución, han producido diversas generaciones de aplicaciones distribuidas.

Le voy a presentar diferentes tipos de aplicaciones distribuidas enmarcados en las generaciones en que han aparecido con lo que mataremos dos pájaros de un solo tiro.

2. Los origines de las primeras aplicaciones Cliente/Servidor.

Recordemos que después de una evolución de la informática departamental basada en Minis y PC's *desconectada* de la informática corporativa, crecieron las necesidades de los datos corporativos de los usuarios de esas aplicaciones periféricas.

Se conjugan en ese momento diferentes factores:

- La comunicación por interfaces de ficheros secuenciales entre ambas informáticas queda desfasada ante las necesidades aparecidas. La presión de los usuarios aumenta.
- Las plataformas PC aumentan espectacularmente en prestaciones y bajan constantemente de precio. A mediados de los 80, los márgenes de las empresas disminuyen o se anulan, los presupuestos se recortan y se hace necesario potenciar el uso de los baratos PC's y de las políticas de desarrollo externo por OutSourcing.
- Los departamentos informáticos, que han llegado incluso a menospreciar los emergentes PC's, se ven obligados a tenerlos en cuenta y estudiarlos. Y del estudio surge el interés.
- Hasta ese momento, los usuarios remotos se han conectado por teleproceso con soluciones punto a punto. Los costes de las comunicaciones se disparan dentro de un entorno económico no muy boyante.

La solución se hace obvia, recurrir a las aplicaciones basadas en PC's en red aprovechando la potencia de ese nuevo entorno. Y **hacer que los usuarios aumenten su participación en el diseño y en la explotación**. Y que con ello se comprometen en el éxito o fracaso.

3. La primera generación de aplicaciones distribuidas C/S.

En este marco surge la primera generación de aplicaciones distribuidas bajo modelo C/S basada en aplicaciones de consulta sobre datos corporativos generados y gestionados a través de las aplicaciones centralizadas. Se conocen simplemente como aplicaciones C/S, obviando, por razones comerciales, el término distribuido.

Sin embargo, las primeras aplicaciones de consulta se montan directamente sobre cada PC de usuario, recibiendo los ficheros corporativos por interfase secuencial que añaden y/o substituyen los nuevos datos sobre los anteriores. Las consultas se suelen realizar con programas específicos realizados para ese usuario con lenguajes convencionales.

Con el aumento de los usuarios interesados y la extensión de las redes, la situación de los datos corporativos replicados cambia. La información corporativa replicada y/o sumarizada se coloca en el servidor de la red de forma que todos los usuarios autorizados puedan utilizarlas. Se potencian las herramientas de consulta estándar que permiten realizar rápidamente consultas no establecidas previamente. Se establecen procedimientos de carga automáticos y la información replicada se actualiza de forma periódica desatendida y automática. Un ejemplo habitual de este tipo de aplicaciones son consultas de resultados de ventas para departamentos de Marketing y Comercial.

Nótese que estas aplicaciones solo tienen de C/S el hecho de compartir un servidor de datos en la red y los recursos de impresión. Sólo en contadas ocasiones se establecen servidores de ficheros entre el HOST y el entorno distribuido para el traspaso automático de la replicación.

Poco a poco aparece un nuevo tipo de aplicaciones, en las delegaciones y los departamentos, consistente en aplicaciones departamentales que registran y modifican datos. Un ejemplo de este tipo de aplicaciones son las de contratación desde las delegaciones comerciales de empresas de servicios. Estas oficinas funcionan de forma autónoma registrando los datos de los nuevos contratos y enviando al final del día las altas, bajas y modificaciones a la central por interfaces locales o remotos.

Se produce en este momento un cambio de requerimientos significativo.

- Necesidad en línea de los datos corporativos, entre ellos los clientes de la empresa.
  - Recordemos la aplicación anterior de contratación. Los datos de los clientes tienen información de una volatilidad tal que la replicación puede quedarse desfasada rápidamente. Imaginemos que un cliente debe recibos y contrata desde otra delegación nuevos servicios. No vamos bien. Como éste, podemos imaginar muchos más casos parecidos. La necesidad de acceder en línea y en caliente a los datos de cliente, stock, etc. se hace imprescindible.
- Registrar los cambios en caliente.
  - Inversamente, la necesidad de tener los cambios registrados inmediatamente en la base de datos central se hace también muy necesaria.

Surgen así los primeros servidores de datos corporativos, que dan origen a las primeras aplicaciones de gestión realmente distribuidas. Y una necesidad nueva,

fundamental, que no había aparecido hasta entonces. Hay cambios en los datos y por tanto **se hace imprescindible garantizar la coherencia e integridad de los datos**. Necesidad que, como cualquiera que ha diseñado aplicaciones informáticas sabe, no es ninguna trivialidad. Y menos en un nuevo entorno distribuido en el cual no había apenas software desarrollado.

Curiosamente, aunque hay movimientos de datos que pueden ser "pinchados" nadie se preocupa inicialmente de la seguridad del transportista.

Hacia el final de esta generación aparecen las primeras aplicaciones avanzadas que sustituyen a los procesos corporativos clásicos del HOST. Son ejemplos normales la mecanización de oficinas bancarias y las aplicaciones de gestión integrada en delegaciones fuera de la cobertura del HOST, y por tanto, desconectadas de él.

La llegada a esta nueva situación es inercial. Repasemos el caso de la oficina bancaria que parte de una situación de teleproceso desde el HOST central. Paulatinamente se incorporan a la oficina PC's que asumen funciones de ofimática y sustituyen a las pantallas de teleproceso mediante emuladores de terminal del HOST para poder trabajar en las aplicaciones centralizadas. Los costes de comunicaciones y la potencia de los PC's suben. Pero los costes de esos PC's bajan.

La solución es inevitable: los PC's deben asumir los procesos departamentales y dejar la conexión al HOST solo para la información en caliente.

Y si falla la conexión en caliente, el sistema debe seguir trabajando. Acaba de nacer el **diseño de consistencia**, aunque muchos diseñadores no se enteraron y por ello algunos sistemas fueron un desastre.

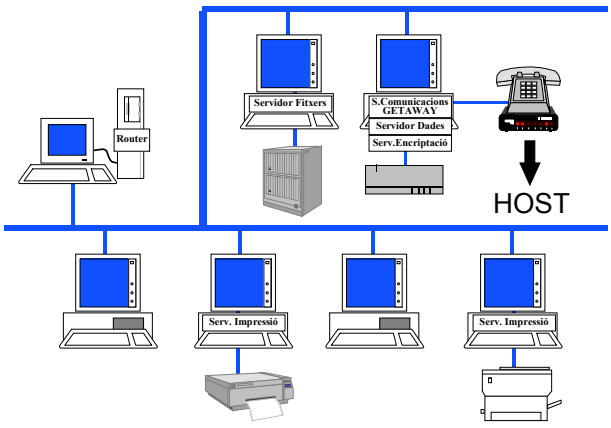


Figura 37. Arquitectura de una oficina bancaria

Respondiendo a esta necesidad, surgen productos especializados por sectores. Por ejemplo, el mercado potencialmente grande de las oficinas bancarias permite desarrollar productos para cubrir las necesidades de esas oficinas. Los productos son potentes pero propietarios de cada fabricante.

Los sectores que no tienen ese potencial de mercado desarrollan aplicaciones propias construyendo todos los servidores de transporte y de sistema, y desarrollando sobre ellos los servidores de aplicación. Son tiempos heroicos pero muy gratificantes para los nuevos profesionales del proceso distribuido C/S.

En poco tiempo, procesos repetitivos como traspaso de ficheros, accesos a las BD's y servicios de impresión se normalizan y se suministran como servicios ya construidos que las emergentes aplicaciones C/S pueden utilizar como componentes a integrar.

4. Evaluación, conclusiones y productos de la primera generación.

La reutilización de servicios se hace evidente y obligada hacia el final de esta generación. Y la necesidad de tipificarlos también. Surgen así los servidores de ficheros, datos e impresión. Estos últimos son de hecho una mezcla de los servicios de impresión del HOST para las impresoras inicialmente al servicio de las aplicaciones de teleproceso y de los servicios de impresión de red que utilizaban las aplicaciones de ofimática. La convivencia crea graves problemas que ahora, con la perspectiva del tiempo, suenan curiosos y sorprendentes. Sobre todo por los quebraderos de cabecera que nos dieron y el tiempo que perdimos con ellos. Quizá el fondo del problema era otro: la lucha entre HOST y PC's. De cualquier forma, en temas como las gestión de servidores de impresión para el uso las impresoras, la red triunfó y los problemas se diluyeron.

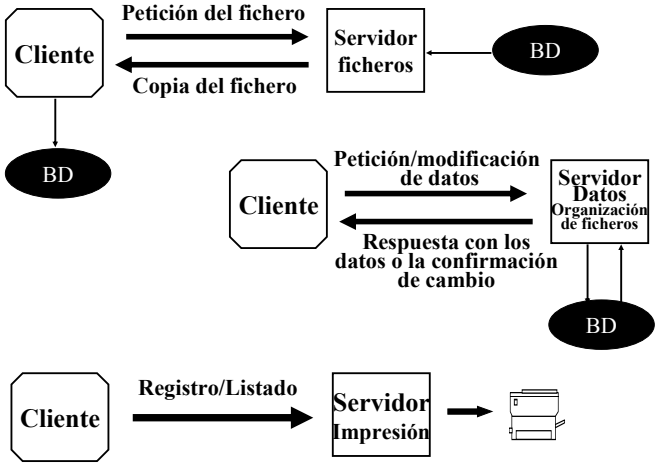


Figura 38. Los servidores de la primera generación.

La evaluación y las conclusiones de la primera generación fueron:

- Se había iniciado la integración imparable de los sistemas.
- Se consolidaron las arquitecturas C/S en red de PC's y HOST-Red-PC. Lo que más adelante se conocerá como arquitecturas C/S basadas en Sistema Operativo.
- Aumentó el interés de los usuarios en participar en la especificación y análisis de las aplicaciones.
- Se estandarizó el transportista sobre las redes y comunicaciones.
- Aumentaron notablemente los servicios de sistema, eso si especializados a cada entorno de sistema operativo. Estos servicios fueron el embrión del Middleware,
- Aparecieron y consolidaron los entornos gráficos.
- Se hizo evidente la necesidad de estándares.
- Se creo un gran caos conceptual y terminológico en los elementos de la arquitectura C/S.
- Se diseñaron aplicaciones de forma anárquica y muchas veces no consistente. Este hecho llevó a muchos fracasos que los reyes de los Mainframe se cuidaron de divulgar.
- Los diseñadores se dividieron de forma muy radical entre *mainframistas* y *cliente/servistas*. Pocos planteaban, yo me apunte a esta lista de solitarios, que los dos sistemas no eran enemigos sino colaboradores.
- Aparecieron grandes problemas para la administración y explotación de los nuevos sistemas distribuidos. Nace así el análisis de administración del sistema distribuido. El problema era de tal magnitud que frenó muchos proyectos.

Aparecen los primeros productos fabricados por las grandes marcas. Son productos que aportan:

- Integración de los entornos HOST de cada fabricantes con los PC's. Además interconexión entre sus propios productos.
- Servidores de datos, ficheros e impresión resueltos.
- Soluciones de comunicaciones.
- Especialización de alto nivel, como por ejemplo, oficinas bancarias.

Estos productos, sin embargo, limitan:

- La transportabilidad ya que estaban muy ligados por API's de bajo nivel a los sistemas operativos propietarios.
- La interconexión entre sistemas de diferentes fabricantes.

5. Aparición de la segunda generación.

Al final de la primera generación sobre los años 91 y 92, aparecen nuevos factores, y sobre todo el concepto de Middleware, que añadidos a la situación resumida al final del apartado anterior, producen el cambio generacional.

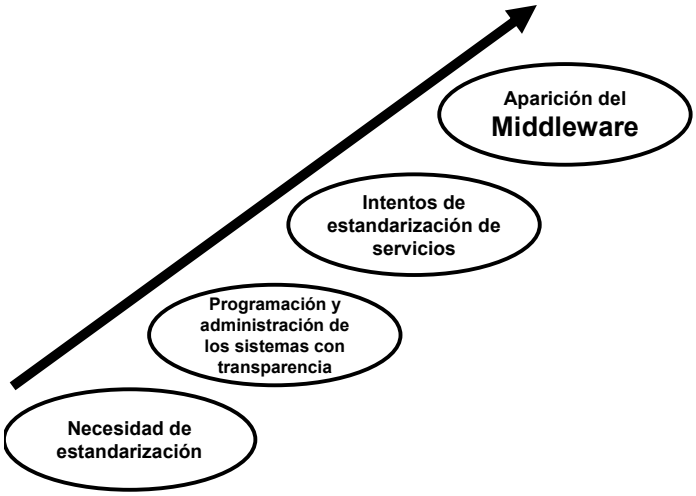


Figura 39. Aparición del Middleware

El nacimiento del Middleware se produce como evolución inevitable de la necesidad de estandarización que había de permitir programar y administrar los sistemas con transparencia mediante la normalización y estandarización de los servicios.

La utilización del concepto de Middleware, el aumento espectacular de prestaciones de los PC's, su bajada continuada de precios, el aumento de la potencia y estandarización de las redes, el aumento acelerado de los estándares "de facto" y las posibilidades de la ofimática, provocaron una verdadera crisis de crecimiento de los sistemas de información en las empresas.

El razonamiento para salir de la crisis parecía obvio, sobre todo para los vendedores de elementos C/S. Había que hacer más muchas cosas en los PC's distribuyéndoles trabajos nuevos o no resueltos eficazmente o a satisfacción de los usuarios hasta ese momento.

Se viven propuestas continuas de estándares para los productos de Middleware que aceleran la evolución de aplicaciones distribuidas a distribuibles.

Se observa claramente la necesidad de encontrar modelos de diseño de aplicaciones C/S basados en arquitecturas de aplicación distribuidas. Y de mejorar y potenciar la conectividad de los sistemas no homogéneos y de diferentes proveedores.

La situación en los años 96 y 97, que yo creo que marca el final de esta generación, era:

- Potenciación de la idea de servicio y servidor: Servidores multimedia, de correo electrónico, presentación, etc.
- Estandarización e integración total de las redes locales incluyendo los principales sistemas operativos. Persistían los problemas para integrar los sistemas remotos.
- Estándares "de facto" además de en entornos de red, en bases de datos (ODBC), integración de productos ofimáticos (OLE), formatos de ficheros gráficos, etc.
- Integración de la ofimática como un componte más de los diseños C/S.
- Integración del paradigma OO en el Middleware y el diseño C/S.
- Existencia en el mercado de gran cantidad de componentes reutilizables.
- La administración de los sistemas C/S estaba bastante resuelta en entornos homogéneos pero persistían los problemas graves para administrar entornos heterogéneos.
- No se concibe una aplicación que interactúe con un usuario sin interfície de usuario gráfica.
- Imposición "de facto" de los productos de Microsoft. Ello tiene una gran ventaja en cuanto a integración, estándares y transportabilidad pero el gran inconveniente de todo monopolio.
- Los sistemas C/S no habían destruido el HOST. Las aplicaciones corporativas de las grandes y medianas compañías seguían en el Mainframe y eran básicamente las mismas de hacía 15 años (o más).
- Aparición con fuerza de los productos integrados tipo SAP como solución global informático/administrativa.
- La inconsistencia y la poca fiabilidad de los productos de software llevan de cabeza a los desarrolladores. En este sentido la huida hacia adelante de los fabricantes es continua: para arreglar un problema hay que subir la versión. Y la nueva versión aporta nuevos problemas.
- Los nuevos productos de software tienen tal necesidad de potencia en los PC's que los parques informáticos de las empresas se quedan obsoletos antes de su periodo de amortización contable.

6. La tercera generación.

Pero las cosas iban a un ritmo acelerado. Entre la primera y segunda generación pasaron casi 10 años. La segunda generación dura menos de cinco años. Y un nuevo cambio ya se produce en el 98.

La aparición de multimedia operativo, el resurgimiento del Mainframe compitiendo en el ratio prestaciones/precio con los PC's, la imposición casi universal de Microsoft y sobre todos ellos la **utilización masiva de Internet** producen el nacimiento de la tercera generación.

La existencia y universalidad de Internet lo cambia todo. Los componentes reutilizables de pueden bajar a través de la red desde cualquier punto del mundo a un coste ridículo para sus prestaciones. Este hecho, unido a que los entornos de programación se potencian extraordinariamente, hacen **que la productividad y calidad de trabajo de los equipos de desarrollo suban espectacularmente**.

Persisten los problemas con los productos de software pero al menos los drivers y revisiones se pueden bajar por Internet.

La utilización de Internet, Intranet y Extranet permite la interconexión remota absolutamente transparente entre elementos de una misma compañía o de diferentes.

Y el estándar de aplicaciones de Internet permite:

- Crear aplicaciones centralizadas utilizando Internet como componente de presentación.
- Utilizar medios de comunicación de Internet como forma de comunicar clientes y servidores en aplicaciones C/S.

Y se produce una reedición de la polémica entre las aplicaciones centralizadas y Cliente/Servidor convencionales. Hay articulistas, que apuntándose a la moda, escriben opiniones lapidarias del estilo: C/S ha muerto. A la opinión se apuntan rápidamente todos los partidarios del diseño centralizado que ven en ello la victoria de sus tesis anti C/S.

Sin embargo y una vez más, personalmente opino que los informáticos estamos de suerte ya que ahora tenemos tres formas de diseño reconocidas agrupadas en dos bloques: centralizada y distribuida C/S en sus dos formas: Cliente/Servidor convencional basada en sistemas operativos y las nueva plataforma basada en Internet. Y además las aplicaciones Internet y las C/S basadas en Sistema Operativo tienen muchas más cosas en común de lo que parece. Si se diseñan, claro está, no si la visión se limita a la programación. Pero esta es ya otra historia de la que ya hemos empezado a hablar anteriormente.

En realidad, como ya hemos dicho, lo que ocurrió fue una generalización del modelo C/S disponiendo de dos formas de implementación: C/S convencional e Internet.

Y, además, los productos integrados de administración aportaron al fin sistemas fiables, cómodos y rentables para administrar los entornos distribuidos.

7. La cuarta generación: la Integración Total de los Negocios.

Y con los primeros años del nuevo siglo ha llegado el inicio de la cuarta generación con la potenciación de la utilización de modelos distribuidos Internet Activos y la generalización de Servicios WEB: la integración total de negocios.

La integración de las tecnologías de la información del concepto de ***Total Business Integration*** **(TBI)** es el objetivo a conseguir en el mundo de los negocios para minimizar costes y aumentar eficacia de gestión. Detrás de este término está el objetivo de hacer interoperables los sistemas de clientes, proveedores y socios.

El reto no es fácil y requiere llegar a pactos que faciliten independencia de las plataformas y de las aplicaciones que soportan los procesos de negocio que se quieren integrar.

Los pasos a seguir son:

7. 1. Integración de los procesos internos.

Para conseguir esta integración global es obvio que hay que asumir no solo la integración externa con terceros sino también la interna de todos los sistemas de información propios.

Hay que crear un escenario suficientemente flexible como para permitir la integración de las nuevas aplicaciones con costes y plazos razonablemente reducidos.

Aquí, los conceptos de diseño e integración de aplicaciones distribuidas en la clave.

7. 2. Integración de las plataformas.

Los conceptos básicos de Middleware distribuido permitirán interconectar de forma transparente las plataformas tanto interna como externamente. Y una reacción más rápida a la evolución tecnológica.

Porque recordemos, que cuando hablamos de integración entre compañías no podemos hablar en ningún caso de unificar plataformas ya que cada una tendrá la suya. Un mundo en que todos trabajáramos con los productos de una sola compañía se parecería mucho más al Mundo Feliz de Owen que al paraíso.

7. 3. Integrar los procesos de negocio interempresariales.

La integración de las aplicaciones de diferentes compañías, y de la semántica de los procesos de negocio que los soportan, necesita de un lenguaje universal común. XML se creó con este objetivo y tiene, pues, un papel básico facilitando la comunicación entre procesos de negocio y, finalmente, entre las personas.

Para conseguir una integración *process to process* se creó XML (eXtensible Mark up Lenguage) como un conjunto de estándares del Consorcio World Wide Web pensado, inicialmente, para integrar procesos de Internet de forma alternativa a EDI (Electronic Data Interchange). La robustez, sencillez y eficacia de la idea lo ha convertido en un estándar para la integración de procesos distribuidos independientemente de si la plataforma es Internet o Sistema Operativo.

La presencia de XML es básica ya que permite avanzar en la integración sin costosas reprogramaciones.

Actuaciones como las de RosetaNet, grupo formado por IBM, Microsoft y American Express entre otros, y BizTalk de Microsoft e IBM, para estandarizar las descripciones, bajo XML, de producto, precios, inventarios, pedidos, facturas, etc.., y de UDDI para crear un estándar bajo XML para disponer un directorio donde registrar los servicios, potencian el camino a seguir.

Obviamente estos estándares pueden utilizarse también en la interconexión de procesos internos. Es más, los nuevos sistemas de información deben primar estos modelos de comunicación, proceso contra proceso, (*Process to Process Communication*) internamente.

7. 4. Interoperatibilidad de Datos.

Para conseguir la operatividad necesaria, se necesitará también que los datos compartidos estén publicados, y por tanto permanentemente actualizados de forma que se eviten al máximo los procesos de replicación e intercambio por interfases.

Finalmente, no obvie que, además de facilitar el acceso a los datos y procesos comunes hay que cerrar los caminos a los restringidos. Así pues, la mejora de los **componentes de identificación y autentificación** de los agentes involucrados y de la seguridad contra ataques externos por las puertas que se abren al exterior es fundamental.

Como ve la idea de utilizar Servicios WEB para interconectar el mundo de los TBI parece, como mínimo, tentadora.

Servicios WEB

1. Introducción.

Dentro del diseño de aplicaciones distribuidas es importante conocer que estructura y funcionalidad proporcionan los Servicios WEB, servicios basados en la red.

En este capitulo vamos a presentarlos. No se trata aquí de hablar de las herramientas y la programación de los servicios sino de su arquitectura y funcionalidad dentro de las aplicaciones distribuidas.

El término Servicio WEB presenta dos acepciones:

- **Funcional** como suministradora de servicios.
- **Tecnológico** como conjunto de tecnologías que permiten este tipo de arquitectura sobre Internet,

Nos interesa aquí, básicamente, el primer aspecto. Si necesita programarlos, consulte bibliografía especializada.

2. Concepto de Servicio WEB.

Un Servicio WEB es un término muy disperso. Hay muchas definiciones y ámbitos que se aplican a este término y, las necesidades de marketing lo han hecho todavía más inconcreto.

Sin embargo, en la raíz del término hay algo tan simple y tan potente como el concepto de servicio que hemos manejado hasta ahora. Y además basado en la independencia y universalidad de la RED.

Abarca, pues, un componente de software que proporciona un servicio bajo plataforma Internet con transparencia de la plataforma, reusabilidad y generalización. Es decir, y en pocas palabras, la esencia del diseño en arquitectura distribuido.

No hacer falta decir que los servicios que se publican y comparten son tanto de datos como de proceso.

Su objetivo es proporcionar interoperatividad sobre una plataforma Internet entre toda clase de aplicaciones y sistemas que se integran de forma transparente siempre que se acojan a los estándares propuestos.

Los servicios WEB se basan en una arquitectura de **objetos distribuidos** en Internet bajo arquitectura C/S, que se comunican por parámetros XML y utilizando como transportista los protocolos estándar de Internet. El concepto en si es el servicio de siempre, pero en un mercado de ámbito y accesibilidad mundial y con un coste de distribución bajo.

Por debajo hay una arquitectura de delegación y distribución de servicios (verá estas arquitecturas de servidores en la segunda parte), en que un servicio invocado puede llamar a otros de forma transparente al peticionario.

La plataforma resultante permite construir aplicaciones distribuidas integrando recursos locales y remotos con total transparencia. Un verdadero proceso distribuido.

En esta plataforma, fabricantes especializados construyen servicios y los publican en Internet.

Permiten la interconexión de plataformas heterogéneas de forma transparente y desconociéndose mutuamente. Es una arquitectura ideal para la integración de procesos de negocio de empresas con sus clientes y proveedores.

Todo ello queda reflejado en la definición de Servicio WEB del Stencil:

> Los Servicios WEB son componentes de software reutilizables, ligeramente acoplados que semánticamente encapsulan funcionalidades discretas que son distribuidas y accesibles a nivel de programación a través de protocolos de Internet.

El concepto "ligeramente acoplado" hace referencia a que en las aplicaciones "clásicas" es muy difícil sustituir en componente por otro. No estoy de acuerdo con el uso de este término. Eso sólo pasa si se han programado mal. Con todo el resto si estoy de acuerdo. Es, además, el concepto de servicio básico para diseñar las aplicaciones distribuidas.

Observe también que "encapsular funcionalidades discretas no es más que otra definición de servicio.

3. Ciclo de vida de los Web Servicies.

El **Proveedor** construye el **Servicio** con el lenguaje y el Middleware necesario.

Define la **Descripción del Servicio** que incluye, con un documento escrito con Servicios WEB Description Language (WDSL):

- Las prestaciones.
- La utilización del servicio por terceros.
- La localización

Publica la oferta del servicio en las páginas amarillas del Universal Description, Discovrey and Integration (UDDI). El fabricante también puede encontrar aquí otros servicios ya creados que le faciliten su trabajo. La **Agenda** UDDI fue creada en septiembre de 2000 por IBM, Ariba y Microsoft y posteriormente se sumaron otros actores como Compaq y SAP.

El usuario final, conocido con el **solicitante** en el argot, **localiza** y **enlaza** el servicio WEB a través de SOAP (Simple Object Access Protocol) mediante un mecanismo de tipo RPC sobre el protocolo HTTP y un intercambio de mensajes XML.

Se baja un applet con el **objeto** que constituye el **Servidor Web,** y que una vez **instanciado** en la plataforma solicitante, le va a conectar con el proveedor

A partir de ese momento, puede empezar a **interactuar** con el servicio, **desinstanciando** el objeto y **desconectándose** cuando ha acabado.

Los Web Servicies Web **no disponen de interfase gráfica** ya que no se han creados para dialogar directamente con los usuarios. Son utilizados como servidores por otros componentes de software del sistema distribuido.

4. Arquitectura Funcional de los servicios WEB.

La arquitectura se basa, pues, en tres tipologías de servicios que se muestran en la figura:

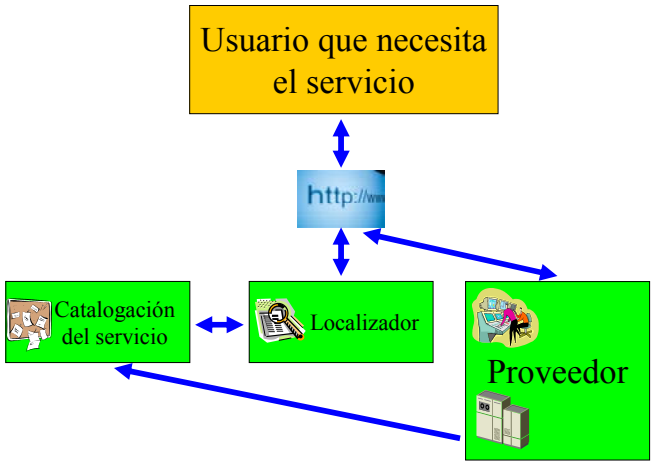


Figura 40. Arquitectura funcional de un Servicio WEB

4. 1. Servicios de Catalogación.

Sirven al proveedor para publicar su servició en la red.

Los aporta la Agencia.

4. 2. Servicios de Localización.

Sirven al usuario para localizar funcionalmente el servicio que necesita.

La localización y descubrimiento del servicio puede ser:

- Estática, navegando el futuro cliente.
- Dinámica en tiempo de diseño o ejecución utilizando un servicio UDDI.

4. 3. Servicios de Utilización

Una vez escocido el servicio y encontrado el proveedor, permiten pedir e instanciar el objeto que debe proporcionar el servicio.

El ámbito de cada servicio ya quedado claramente definido en el apartado anterior.

5. Una visión genérica de los servicios de publicación.

Los servicios UDDI disponen de:

- Paginas blancas para la localización por la identificación: nombre, dirección, etc...
- Paginas amarillas catalogación por tipología y categoría del servicio.
- Paginas verdes para la descripción técnica.

La publicación en UDDI sigue los pasos siguientes puede hacerse por programa o por WEB.

Una descripción de servicio WSDL es un documento XML que contiene:

- Qué hace el servicio a través de la descripción de sus métodos.
- Como se accede a él a través del formato de datos y parámetros.
- Dónde está localizado a través de un URL.

6. Arquitectura Tecnológica de los servicios WEB.

La arquitectura de los servicios WEB es una Arquitectura Orientada a Servicios (*SOA - Service Oriented Architecture* de la que hablaremos al inicio de la segunda parte) formada por una colección de servicios que se interconectan entre ellos para conseguir una determinada actividad.

Los servicios de localización y comunicación son uno más de la lista disponible.

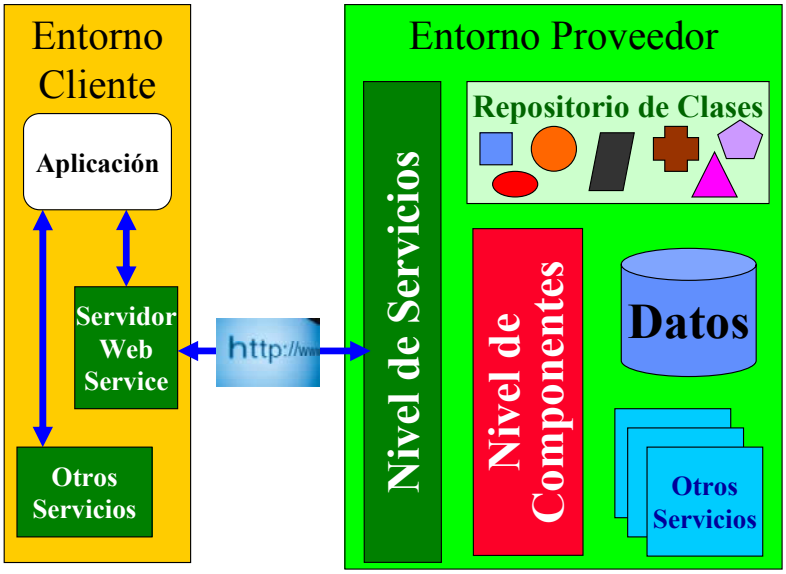


Figura 41. Arquitectura tecnológica de un Servicio WEB

El funcionamiento es el esperable dentro de una arquitectura distribuida como las que ya hemos presentado.

- El componente de la aplicación cliente diseñado para interactuar en el servicio, no necesariamente el módulo principal, pide el servicio al proveedor a través de un enlace de red.
- El nivel de servicio del proveedor le envía un objeto que se instancia en el entorno de la aplicación y se constituye en un servidor WEB. Observe que de esta forma dispone siempre de la última versión del servicio.
- El componente cliente empieza a utilizar el servicio con el conocimiento de sus métodos que le ha proporcionado la descripción publicada en el documento WSDL.
- A partir de ese momento, la aplicación utiliza el servicio como uno más de sus servicios locales.
- Cuando se pide al servidor WEB que trabaje, este realiza un traspaso de responsabilidad, servicios de pasarela o delegación que veremos en la segunda parte, al nivel de servicio del proveedor que lo traspasa a su vez a los componentes, servidores, de su área local que proporcionan el resultado pedido utilizando sus servicios de proceso y datos locales y/o remotos ya que puede utilizar otros Servicios WEB como parte del proceso.
- La comunicación entre cliente y servidor se realiza a través de protocolos comunes a Internet.
- Cuando la necesidad del servicio acaba, el objeto se destruye.

Nótese que con esta arquitectura tecnológica, y como se ha dicho antes, **el cliente dispone siempre de la última versión del servicio, tanto en depuración de errores como en nuevas prestaciones.**

Existen otros componentes en esta arquitectura tecnológica que deben tenerse también en cuenta.

- Autentificación y seguridad.
- Integración en el Middleware.
- Calidad de servicio.
- Administración.

Existen otros factores a considerar en el diseño de este tipo de servicio. Uno de ellos es la granularidad. El intercambio de datos debe ser siempre lo más amplio posible lo que facilitará que muchos clientes puedan disponer en tiempo razonables del servicio.

Otro factor a tener en cuenta es el respeto a la interfase de parámetros en los cambios de versiones.

7. Roles de un Servicio WEB en un sistema distribuido.

> Un Servicio WEB es un servicio. Punto. No hace falta añadir nada más en cuanto a sus roles en un sistema distribuido.

Sin embargo es tradicional que se hable específicamente de sus roles funcionales. No me importa. Extiéndalo a cualquier servicio.

- Unificación de parámetros de servicios equivalentes con heterogeneidad de formatos.
- Integración de procesos de negocio entre dos entornos que se desconocen tecnológicamente.
- Envolvente de aplicaciones heredadas.
- Agente en un WorkFlow. Se utiliza el estándar WSFL para la descripción de este tipo de proceso.
- Etc.

Aquí me paro. **Cualquier rol de un servicio le es propio**.

8. EDI y Servicio WEB.

Desde hace tiempo EDI (Electronic Data Interchange) se utiliza como plataforma de comunicación entre sistemas heterogéneos, ya sea por plataforma o por desconocimiento. Desde hace años, clientes y proveedores se han comunicado entre ellos usándolo a través de un Centro Servidor EDI. Existe versión de EDI para Internet.

Los defensores de los Servicios WEB los proclaman como alternativa al EDI. La polémica está servida de nuevo. EDI es más caro y pesado pero más seguro que los Servicios WEB que son justo lo contrario.

Si se ve en la necesidad deberá elegir y estará muy condicionado por el sector económico por el que se mueva su empresa. En aquel momento infórmese y decida según su caso.

9. Plataforma de desarrollo de Servicios WEB: Java de SUN versus .Net de Microsoft.

J2EE basada en Java y .NET no son más que dos tecnologías para construir servicios WEB.

.NET es la plataforma de Microsoft. Es un conjunto completo de herramientas basadas en XML, XSML y el resto de estándares que proporcionan recursos para desarrollar, utilizar y administrar servicios WEB.

Proporciona un lenguaje de descripción de clases que permite crearlas en los lenguajes de programación más habituales y utilizarlas después de forma transparente en el entorno .NET.

Aporta también un Framework que marca una guía de cómo construir aplicaciones distribuidas para dar servicios WEB.

J2EE es la plataforma de SUN basada en Java, estándar de facto del mundo Internet. J2EE potencia y complementa Java con una biblioteca de herramientas, funciones y servicios como JDBC para acceso ODBC a bases de datos, EJB, un modelo de servidor para localizar los servicios creados, Java IDL para interactuar con CORBA, uso de XML; etc..

J2EE es, más que un producto, una propuesta de estándares a los que los fabricantes deben adherirse. J2EE implica a diversas empresas proveedoras de Middleware cooperando entre sí, siempre con Java como lenguaje de desarrollo.

Orientación: Net está centrada en Windows y es independiente del lenguaje. J2EE está centrada sobre Java y es independiente de la plataforma.

Hay, sin embargo, un denominador común: la convergencia en XML.

¿Por cual decidirse? Depende de su necesidad y la historia previa de su sistema de información. Sin embargo, la convivencia puede ser una buena opción.

Influencia de una arquitectura distribuida en la construcción de aplicaciones.

1. Introducción.

Conocemos ya las características básicas de una arquitectura distribuida. Es el momento de revisar en que fases del desarrollo y la gestión de un sistema de información influye.

2. Etapas del ciclo de vida donde influye la arquitectura distribuida.

En la figura se muestra el clásico ciclo de vida en cascada. ***Una aplicación distribuida nunca debe desarrollarse según este modelo en cascada***. Particularmente creo que ni distribuida ni ninguna.

He elegido este modelo para presentarle los puntos del ciclo de vida donde influye la arquitectura distribuida por lo claro que quedan en él las etapas de desarrollo.

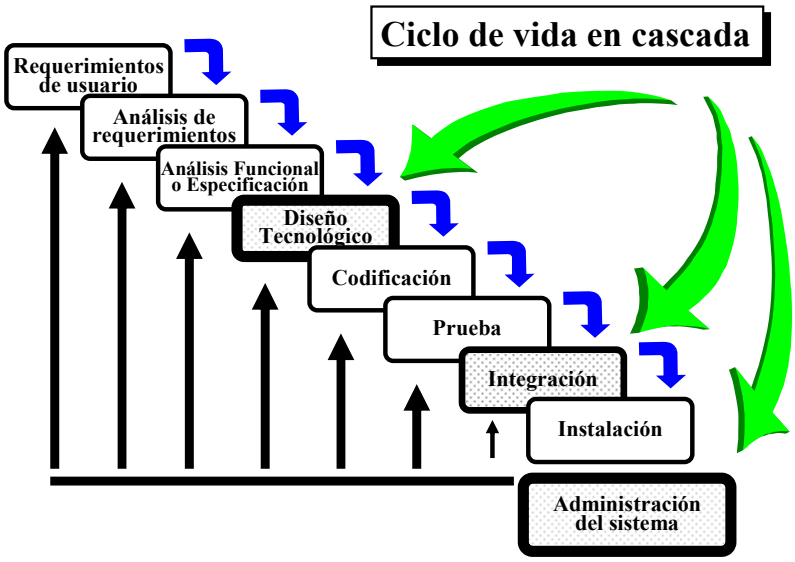


Figura 42. Ciclo de vida en Cascada.

El ciclo de vida en cascada se inicia con una petición de usuario reflejada en unos *Requerimientos de Usuario*, verbales o escritos, pero nunca muy precisos.

A partir de aquí el informático elabora una *Análisis de Requerimientos* que realiza una aproximación a la viabilidad del proyecto, en plazos, costes y recuperación de la inversión. En esta fase suele haber ya una primera valoración del nivel de distribución que tendrá la aplicación.

A partir de aquí de desarrolla un *Análisis Funcional o Especificación* que ha de reflejar de forma completa y precisa que espera el usuario de la aplicación. En los sucesivo, nos referiremos a esta etapa por Análisis Funcional.

Todos los que trabajamos en diseño de aplicaciones distribuidas coincidimos en que estas tres primeras etapas son independientes del hecho de que después la aplicación puede ser o no distribuida. Es más, personalmente creo que pensar en la distribución en fase de funcional es un gravísimo error que puede llevar a adelantar decisiones de arquitectura distribuida, e incluso del modelo de distribución, antes de tener toda la información que proporciona un funcional completo. Y estas decisiones prematuras llevan en muchísimos casos a errores graves que penalizan para siempre la aplicación.

Reconozco con Vd. que cuando se tiene experiencia se hace difícil no pensar en fase de funcional como se montará alguna parte de la arquitectura distribuida. Pero quédese ahí, en una intuición. Deje las decisiones para el momento oportuno, el tecnológico.

Una vez redactado el funcional y pactado con el usuario, empieza la etapa de diseño tecnológico. Este ciclo se inicia con el ***Diseño de la Arquitectura del Sistema***. Es aquí donde impacta de lleno el hecho de que la arquitectura de aplicación escogida es distribuida. La idea fundamental es que habrá que incorporar unas etapas adicionales de diseño para contemplar este hecho.

La siguiente etapa del tecnológico es la *Programación y Codificación* de los algoritmos necesarios. Esta etapa vuelve a ser independiente de la arquitectura distribuida. La influencia de la distribución ya se ha pactado en la etapa anterior. Una vez construido, cada programa se *Prueba* individualmente y se integra en su bloque operativo.

Una vez probados todos los programas por el equipo de desarrollo se realiza la ***Integración*** sobre la plataforma definitiva. Los recursos distribuidos se localizan en esa plataforma y hay que comprobar el buen funcionamiento integrado de los componentes distribuidos. Esta etapa, que en una aplicación no distribuida suele ser un apéndice de la etapa de Prueba, **es en un sistema distribuido fundamental**. Y marcará el éxito o el fracaso de la aplicación. Hay que comprobar rendimientos, tolerancia a fallos, procesos de recuperación de servicios, vías alternativas, etc. Dicho de otra forma, hay que verificar completamente el **diseño de consistencia**. Es decir todos aquellos elementos que hacen los diseños distribuidos diferentes y que hemos citado e iremos citando a lo largo del libro. En el capitulo siguiente ya le daré una idea general de cuales son.

Finalmente se produce la *Instalación y arranque* de la aplicación que si la etapa de integración se ha desarrollado correctamente, no debería ser básicamente diferente al de una aplicación no distribuida.

El otro punto básicamente diferencial de una aplicación distribuida es la **Administración del Sistema Distribuido**, situación en que las aplicaciones distribuidas están en clara desventaja sobre las centralizadas.

El tema de la administración es suficientemente importante como para que sea motivo de estudio exhaustivo más adelante.

3. Áreas temáticas de formación para desarrollar aplicaciones distribuidas.

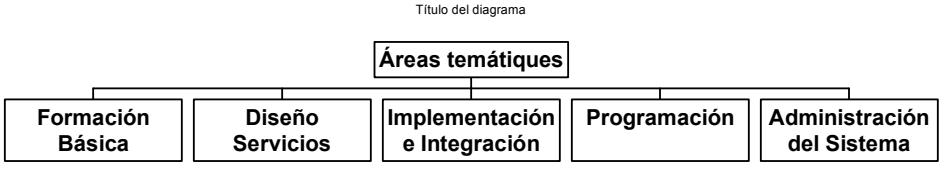


Figura 43. Áreas temáticas de formación para diseñar sistemas distribuidos.

Hay cinco grandes áreas temáticas que hay que referenciar para situarse en el entorno de desarrollo y administración de sistemas distribuidos: Formación násica, Diseño, Implementación e Integración, Programación y Administración del Sistema.

A continuación se explica que hay que conocer de cada una de estas áreas.

4. Formación básica.

Figura 44. Áreas de la formación básica

Cualquier profesional que desee diseñar aplicaciones distribuidas deberá disponer de conocimientos en cuatro áreas:

4. 1. Arquitectura Cliente/Servidor.

Los conocimientos es esta área deberán ser, evidentemente, completos. Ha de saber como cambia el diseño y la administración del sistema. Y que ventajas e inconvenientes tienen las aplicaciones distribuidas por C/S sobre otras. Todo ello le permitirá tomar las decisiones de viabilidad, estrategia, diseño y administración que convengan en cada caso.

4. 2. Conectividad.

En esta área se deben conocer las posibilidades de conectividad de la plataforma actual y de la oferta de mercado. La aplicación puede necesitar en el momento del diseño distribuido modificar o ampliar la plataforma y por esa razón hay que estar al día.

No necesita ser un experto. Ya hemos comentado que conectividad es una especialización diferente a diseñar. Cuando necesite información especializada consulte con un experto, interno o externo.

Evidentemente no necesita para nada saber como se administra una red, ni un sistema de seguridad. Vd. debe conocer solo que servicios le reportan para desarrollo y administración. Y como usarlos. Podrá utilizar los primeros para desarrollar y deberá cubrir con su aplicación las posibles carencias de los segundos.

Obtener esta formación básica es difícil ya que hay muchos libros sobre como administrar conectividad y pocos sobre explicaciones funcionales de los servicios que da. Y prácticamente ninguno que compare ventajas e inconvenientes de las diferentes soluciones.

Yo, que ya he confesado antes que no soy experto es este tema, he empleado un sistema que no se si le será válido. He preguntado mucho a los que realmente saben por formación y, no se olvide, por experiencia (fundamental). He asistido a presentaciones comerciales, donde se habla más de posibilidades que de administración. En este caso no se olviden de dividir las posibilidades fantásticas que oirá por el coeficiente de incremento del Dpto. Comercial o de Marketing que le haga la presentación. Y he intentado aprender cuanto he podido de la experiencia.

4. 3. Internet.

En un capítulo anterior ya hemos tratado las reacciones entre C/S y Internet y como éste es un modelo C/S disponible para implementar aplicaciones distribuidas.

Un buen diseñador de aplicaciones distribuidas debe conocer las posibilidades de las aplicaciones Internet ya que gran parte de las aplicaciones distribuidas pueden y deben implementarse hoy día en Internet. Y al revés, no desviar parte de la aplicación distribuida a Internet si se implementaría mejor con C/S basado en Sistemas Operativos. Pero cuidado, las modas mandan y condicionan.

El conocimiento de Internet debe ser exhaustivo en posibilidades, no en herramientas. El paralelismo es el mismo que confundir diseño con lenguaje de programación. Y cuidado, que aquí la confusión es todavía más fácil.

Y, por favor, no caiga en el error sobre el que le avise al hablar de conectividad. Debe conocer posibilidades y funcionalidad de Internet, no administración física ni diseño artístico.

4. 4. Servicios disponibles en el mercado.

Cada vez más se hace necesario estar al día de los servicios disponibles en el mercado y que pueden usarse con solo contratarlos.

En este tema este muy atento a la evolución del mercado de Servicios WEB.

4. 5. Orientación a Objetos.

Se puede hacer C/S e Internet sin saber OO. Sin embargo, conocer la tecnología ayuda muchísimo. Más adelante plantearemos una reflexión sobre las dos tecnologías.

5. Diseño i servicios.

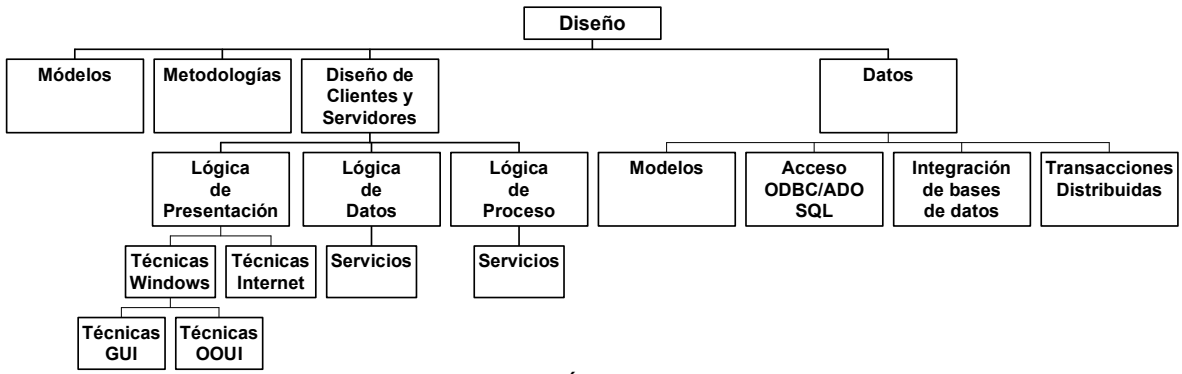


Figura 45. Área temática de diseño

La influencia de la distribución en el diseño se centra en cuatro grandes temas:

5. 1. Modelos.

Como en todas las actividades de la Ingeniería, en el diseño informático en general, y en el C/S en particular, disponer de modelos para analizar los problemas, asociarlos a uno de esos modelos y aplicarles el esquema de solución asociado al modelo es fundamental.

Más adelante presentaremos y analizaremos los diferentes modelos del mundo C/S.

5. 2. Metodologías.

Las metodologías nos darán la pauta para resolver los problemas. Dentro de este libro encontrará las metodologías que le propongo para diseñar las aplicaciones distribuidas.

5. 3. Diseño de clientes y servidores.

Como programar los programas clientes y servidores no se diferencia mucho del resto de programas del mundo de la informática, sólo hay que añadir criterios de distribución de funciones y recomendaciones que encontrará más adelante en capítulos especializados.

Existe una lógica de presentación, de datos y de proceso que hay que repartir entre clientes y servidores. La lógica de presentación irá siempre en los clientes pero las lógicas de datos y proceso se repartirán entre clientes y servidores. Más adelante trataré el tema en profundidad.

5. 4. Diseño de servicios.

Obviamente, hablar de servicios es equivalente a hablar de servidores.

5. 5. Datos.

La organización, integración y gestión de datos es uno de los temas más duros de esta tecnología y uno de sus más graves inconvenientes si no se pueden centralizar los datos ya que es obvio que los datos centralizados se pueden gestionar infinitamente mejor que los distribuidos.

Al hablar de datos tendremos que definir también modelos, hablar del acceso, hoy unificado en ODBC-ADO-JDBC-SQL, hablar del problema de la integración de datos e introducir el diseño por transacciones distribuidas que, en la práctica se usa en contadísimas ocasiones.

6. Implementación e Integración.

En cuanto a la implementación e integración de las aplicaciones distribuidas en general y C/S en particular conviene formarse en tres apartados:

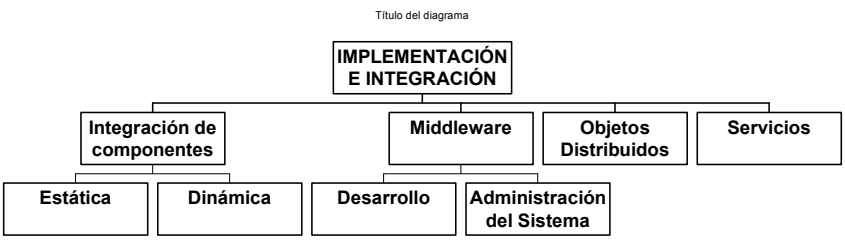


Figura 46. Área de Implementación e Integración

6. 1. Integración de componentes.

Tendrá dos formas de hacerlo.

6.1.1. Estática.

En el momento de linkar el programa.

6.1.2. Dinámica.

Arrancando el componente en fase de ejecución. En este caso debe conocer las posibilidades del Middleware para integrar, catalogar, localizar, arrancar y comunicar componentes.

6. 2. Middleware.

Sin comentarios. Crear aplicaciones distribuidas es imposible sin conocer perfectamente las posibilidades de Desarrollo y Administración que le proporciona el Middleware.

Debe tener los conocimientos genéricos sobre las posibilidades del Middleware y los específicos del Middleware de su instalación.

6. 3. Objetos Distribuidos.

Este aspecto de su formación es optativo pero cada vez más importante por su utilización sobre TCP/IP que hace transparente C/S convencional y Internet. Sólo profundice en el si es un experto en OO y dispone de una plataforma J2EE, .NET-DCOM o CORBA operativa o si quiere utilizar este tipo de recursos.

6. 4. Servicios.

Implementar e intregrar por servicions supone aprovechar las plataformas de Middleware disponibles, principalmente Internet i Web Services.

7. Programación.

A

e

f

e

s

d

e

p

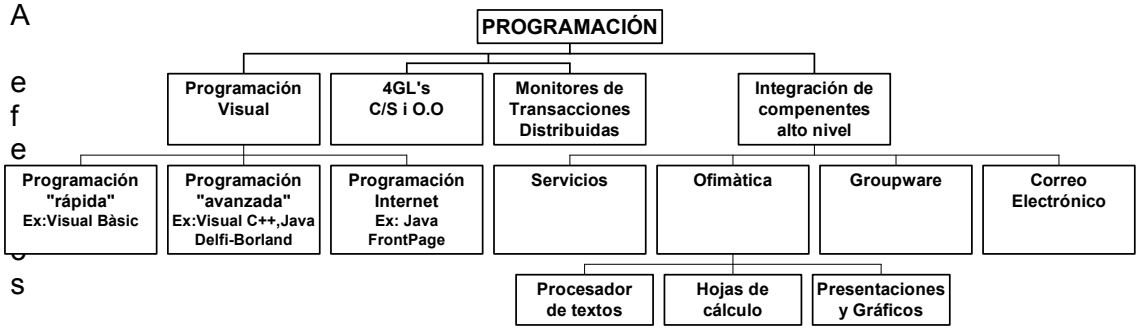


Figura 47. Área de Programación

En cuanto a la programación, veremos que hay dos tipos básicos de aplicaciones distribuidas:

- Aplicaciones de Desarrollo Rápido.
- Aplicaciones avanzadas.

Más adelante encontrará ampliamente explicado el concepto de ambos tipos de aplicaciones.

De Internet y C/S ya se ha hablado en un capítulo anterior.

Es mi opinión, que será compartida o no por Vd. según cual sea su lenguaje de programación favorito, es que hay lenguajes más adaptados a cada uno de los dos tipos de aplicaciones.

Visual Basic es un buen lenguaje para montar aplicaciones de desarrollo rápido. DELPHI lo es para las avanzadas. Java es el lenguaje natural de Internet que además permite programar desde complejos servicios a clientes y terminales móviles. Además, la arquitectura J2EE basada en Java permite trabajar de forma única con los dos tipos de aplicaciones C/S, Internet y basadas en sistema operativo. Y los 4GL's OO y C/S permiten, además de las técnicas propias de los 4GL los tres tipos de aplicaciones.

Sin embargo, es bien cierto que cada vez hay menos diferencias entre los lenguajes actuales y que todos permiten montar todo. Elija el que prefiera. No es una decisión crítica. Su única restricción debería ser trabajar con lenguajes que permitan compartir componentes y/o rutinas independientemente de si se han desarrollado en uno u otro lenguaje.

Las herramientas de los Monitores de Transacciones Distribuidas son potentes pero no se utilizan en la práctica.

Y del uso de Componentes de Alto Nivel se hablará, por su importancia, más adelante.

Orientación a Objetos y Cliente/Servidor

1. Introducción.

Se puede trabajar en Cliente/Servidor e Internet sin aplicar técnicas de Orientación a Objetos.

Sin embargo conocer, como mínimo, los fundamentos de OO y aplicarlos a la programación de aplicaciones distribuidas es muy recomendable para aprovechar las posibilidades de encapsulamiento y reutilización de OO. Y si no se conoce OO aplique como mínimo técnicas de Tipos Abstractos de Datos (TAD).

Si no conoce estas técnicas consulte bibliografía sobre el tema. En la segunda parte le haré una introducción rápida para que pueda segur el diseño si no tiene formación en OO ni TAD.

A continuación se incluyen algunos comentarios sobre ambas, no con la idea de explicar que son, sino en la de ver como utilizarlas en Cliente/Servidor, englobando en este término tanto las aplicaciones distribuidas basadas en sistema operativo como en Internet.

En orden lógico de exposición, quizás este apartado debería ir más adelante, dentro de los capítulos de implementación. Particularmente, creo que es mejor localizarlo este momento para poder hacer referencia a estas técnicas cuando convenga.

2. Cliente/Servidor y Orientación a Objetos.

Quiero hacer una consideración previa que quizás escandalice a muchos expertos en Orientación a Objetos. Soy un firme convencido de que se puede diseñar “convencionalmente” y programar OO para aprovechar así todas las ventajas que el paradigma aporta para programar. En ese convencimiento escribo.

Las cualidades deseables en implementación de piezas forman parte consustancial de las técnicas OO.:

- Encapsulamiento.
- Polimorfismo.
- Transparencia.
- Reusabilidad.

Además, OO aporta otra cualidad muy útil, la herencia, que no se necesaria en el concepto básico de pieza, pero que puede ser de gran utilidad si se domina. El concepto de pieza lo introduciré en la segunda parte del libro. De cualquier forma le adelanto que una pieza no será nada más que un componte con datos y proceso encapsulados en su interior.

Además, otras poderosas razones llevan a la sinergia entre C/S y OO:

- Es un hecho que las herramientas C/S, Internet y los productos de Middleware actuales son OO.

- Las técnicas de presentación gráfica (súper utilizadas en aplicaciones distribuidas) son OO.
- La presencia en los entornos de Middleware de un ORB (*Object Request Brokers*) para la gestión de objetos distribuidos.
- La utilización de técnicas de prototipaje en los entornos C/S favorece también las técnicas OO.

Figura 48. Alianza C/S - OO.

Evidentemente se puede hacer C/S sin OO, e incluso programar en Java sin OO, pero todo el entorno favorece que OO y C/S se combinen estupendamente bien.

En particular, las técnicas de Objetos Distribuidos son interesantísimas en la implementación de aplicaciones distribuidas.

Los modelos de objetos distribuidos, gestionados mediante un ORB permiten, en esencia, diseñar y/o programar OO y después distribuir los objetos por diferentes máquinas o plataformas de forma transparente. El modelo proporciona recursos de programación, de integración y de administración bajo un estándar.

Como ya hemos comentado otras antes, hay varias propuestas de modelos OO distribuidos.

- **J2EE,** en el mundo de Java.
- **CORBA**, promovido por OMG y empresas como IBM, SUN, HP y apuesta de los entornos JAVA.
- **.Net y DCOM**, promovido por Microsoft.

Una desventaja, nada despreciable, es la inversión en formación que se ha de hacer si el equipo de desarrollo no está preparado en técnicas OO.

Consulte bibliografía especializada si desea más información sobre todos estos temas.

Para seguir este documento no necesitará mucho más. Y cuando lo necesite, yo le ayudaré.

3. Tipos Abstractos de Datos (TAD).

En una aproximación rápida, un TAD es un objeto sin polimorfismo ni herencia (y que me perdonen los puristas).

Cada TAD define sus tipos de datos y las rutinas de gestión a través de los parámetros de la cabecera.

Así, la especificación del TAD tiene tres partes:

- Definición de tipo o tipos base asociados, que pueden quedar ocultos o no a los programas usuarios.
- La declaración de rutinas para la gestión del TAD. En entornos C/S los servicios se encapsulan en estas rutinas.
- La implementación de las rutinas, que quedan ocultas al programa que las utiliza.

Evidentemente implementar una pieza por un TAD es, conceptualmente, trivial.

Componentes de Alto Nivel: Ofimática y Groupware

1. Introducción.

La utilización de componentes ya construidos es fundamental dentro de la construcción de aplicaciones C/S.

Además de otras muchas, hay dos fuentes muy útiles de componentes:

- ❍ Ofimática.
- ❍ Groupware.

No pretendo aquí presentar exhaustivamente ambas fuentes de componentes de alto nivel. Solo pretendo hacer algunas reflexiones y proponer algunos ejemplos para hacerle ver la importancia de utilizarlos. Y estimular su imaginación.

2. ¿Qué es un componente de alto nivel?

> Cuando hable de Componente de Alto Nivel me referiré a programas que se usan por si solos pero que las aplicaciones distribuidas pueden utilizar para cubrir parte de sus funciones.

Por ejemplo, un procesador de textos es un componente de alto nivel. Sin comentarios sobre su uso individual. Pero puede usarse también como un servidor de impresión de facturas si la aplicación, distribuida o no, le delega la función de impresión con formato.

Existen diferentes formas de utilizar componentes de alto nivel. La utilización de uno u otro sistema será una decisión del diseñador en función de las necesidades de su aplicación.

2. 1. Como servicio.

El componente de utiliza como un servicio más dentro de la aplicación distribuida.

2.1.1. Comunicación con filosofía de agente.

El componente de alto nivel asume el rol de agente. Básicamente puede iniciarse de dos formas:

- Por interfase.
  - El cliente o el servidor le pasan al componente un fichero con los parámetros del servicio que le interesa.
- Por comunicación asíncrona desacoplada, de la que hablaremos más delante.

El componente de alto nivel se arranca por el usuario a voluntad. La aplicación distribuida no realiza ninguna coordinación de ambos trabajos.

2.1.2. Integración como servidor.

Se diferencia del anterior en que los parámetros del servicio solicitado al componente de alto nivel se le traspasan mediante una petición de servicio, normalmente síncrono, utilizando uno de los mecanismos de comunicación Cliente/Servidor de los que hablaremos más adelante.

2. 2. Como estereotipo.

El componente de alto nivel asume un rol como parte de la arquitectura distribuida que interactúa directamente con el usuario.

Por ejemplo, un ERP permite entrar o validar apuntes contables sobre una hoja de cálculo. Este recurso puede servir para validar las cifras de venta por toda la red de agencias de una empresa. EXCEL ha asumido el estereotipo funcional de *validador*.

Nota: La integración por estereotipos es una forma de integrar las funciones clientes que se verá al final de la segunda parte.

Este proceso tendrá normalmente varios pasos:

- Generación desde el ERP de la Hoja de Cálculo.
- Envío de la Hoja de Cálculo al Usuario, adosada a un mail, por ejemplo.
- Relleno de la hoja por el usuario usando directamente la Hoja de Cálculo.
- Devolución al centro del ERP.
- Validación e Integración con los datos del ERP.

Observe que este proceso, en apariencia simple, si se usa con muchos usuarios simultáneamente, necesita como mínimo:

- Un proceso automático de generación y envío.
- Un proceso desatendido de recepción.
- Por proceso automático y muchas veces desatendido de incorporación
- Un sistema de control de situación de cada Hoja de Cálculo: enviada, recibida, incorporada, etc.
- Quizás un proceso de reclamación automática.

En fin, todo un módulo dentro de la aplicación distribuida. Si cae en el error de no diseñarlo está perdido: el coste de la mano de obra necesaria para llevar el control y el agobio de tiempo le llevará al fracaso.

3. Ofimática.

Vd. es, seguro, un usuario habitual de las suites de ofimática. Procesadores de texto, hojas de cálculo, presentaciones, bases de datos de usuario (tipo ACCESS), paquetes de gráficos, editores de imágenes, etc., son usados habitualmente por informáticos y no informáticos.

Y en entre las suites de informática la más ampliamente utilizada en Microsoft Office con sus populares Word, Excel, PowerPoint, Access, etc. Y también el Open Office

del mundo del software libre muy utilizado en Linux aunque dispone también de versión en Windows.

Todas las funciones de estos paquetes puede ejecutarse y pedirse desde otros programas y llamadas a través de API's.

Por su naturaleza, la mayor parte de las aplicaciones distribuidas se ejecutan sobre plataformas donde estos paquetes están presentes y además, en muchos casos, en todas las máquinas clientes.

Paralelamente, las aplicaciones distribuidas necesitarán en muchísimas ocasiones funciones naturales en estos paquetes. ¿Por qué no utilizarlas?

Déjeme proponerlo algunos ejemplos adicionales.

Una aplicación de atención al cliente en una tienda de un aeropuerto debe imprimir facturas en diferentes idiomas. Sin un procesador de textos, la aplicación debería dar herramientas para proporcionar los textos en diferentes idiomas: es decir, habría de incluir un mini-procesador de textos.

Si programa este recurso dentro de su aplicación no alcanzará nunca las prestaciones más pequeñas de un Word. Y además, sería un esfuerzo inútil de desarrollo que encarecería plazos y costes. Deje al Word lo que es del Word y al programa a medida lo que es del programa a medida. Prepare los datos en el programa y llame al Word pasándoselos. Word hará su función perfectamente.

Otro uso en el que Word facilita la vida es el diseño de impresos con formato variable. Basta que los defina a través de Word y que desde la llamada del programa especifique que formulario necesita en cada caso.

Además no menosprecie una cosa fundamental. Respetando las pocas reglas que Vd. dará, los usuarios pueden mantener ellos mismos los formatos de sus impresos. ¡El sueño de nuestros usuarios y permite, además, deshacernos de una de nuestras pesadillas!

Excel puede utilizarse como herramienta para gestionar información de investigación. Por ejemplo, imagine que Vd. tiene una muy buena estadística de ventas de productos por locales, con un buen ámbito de selección y que permite su exportación a Excel. Un usuario de Marketing lanza una campaña de una oferta y quiere hacer un seguimiento.

Realizar programas específicos para estos análisis es caro en relación con el poco tiempo que se utilizarán. Y cuando se acaba la campaña ya no se necesitan más. Si existe la posibilidad de exportación a Excel, nuestro usuario la utilizará y desde Excel manipulará y elaborará los datos según sus necesidades.

Como buen informático pensará que es mejor usar Access que Excel para cubrir estas necesidades. Se olvida de que el usuario no suele ser un experto en Access. En el mejor de los casos es un experto en Excel pero no conoce los mínimos de lógica de programación ni de diseño en entornos gráficos para hacer algo en Access más halla de utilizar formularios predefinidos, que seguro, se le quedarán cortos. Esta es la **gran ventaja de Excel: todo el mundo sabe usarlo** y se pueden sacar resultados desde el primer momento. Si habitúa y forma su organización en el uso de hojas de cálculo, ese tipo de análisis le sacará de encima una pila de problemas y presiones.

Usar las posibilidades de Excel y/o PowerPoint para incrustar gráficos en las pantallas no debe tampoco despreciarse. Si programa los gráficos, cada vez que se le pida un cambio deberá intervenir. Si Vd. pasa los datos y el Excel realiza el gráfico, el usuario podrá cambiarlo y manipularlo a su voluntad.

Los ejemplos de utilización son tantos que podrían construir un libro por si sólo.

Desgraciadamente hay muchísimos libros de ofimática de usuarios, más o menos avanzados, pero poquísimos de como usarlos desde programación y, yo al menos no conozco ninguno en castellano, de que funciones, incluyendo ejemplos, pueden cubrir en un diseño de aplicación.

4. Groupware.

El nombre de Groupware engloba una colección de tecnologías de alto nivel que permiten procesos complejos centrados alrededor de actividades humanas desarrolladas en colaboración y a través de ordenadores.

Están basadas en técnicas C/S y multimedia y apoyados en sistemas distribuidos cercanos a los usuarios y con niveles de interoperatividad en las conexiones entre esos usuarios. El volumen de los datos involucrados obligan, para ser eficaces, altas velocidades de conexión. Este hecho puede dificultar las actividades remotas.

Groupware fue el resultado de la evolución y concentración de productos en la línea de “trabajo en grupo”, “trabajo en colaboración”, “correo electrónico”, “vídeo conferencias”, “trabajo cooperativo soportado por ordenador”, etc..

Las herramientas de Groupware no tienen unos estándares bien definidos y son de muy difícil clasificación. No pretendo aquí realizar nada de ese estilo.

Solamente quiero, si Vd. no las conoce ya, hablar de los productos genéricos y explicarle los que, a mi juicio, pueden serle útiles en su diseño de aplicaciones distribuidas.

Si conoce el tema, sáltese este apartado.

4. 1. Gestión de documentos multimedia.

La unidad básica de trabajo es el documento.

Cada documento se le asigna un propietario(s), un autor(es), un tema(as), unas palabras claves de documentación, los componentes multimedia que se incluyen, los productos de software para gestionar cada componente, las tareas coordinadas para su creación y mantenimiento, etc..

Los documentos se almacenan en servidores de bases de datos, habitualmente en formatos de tipo BLOB (el formato es desconocido para la BD y es responsabilidad de los programas de gestión). La gestión se realiza desde las máquinas clientes. También se utilizan ficheros XML.

Los productos de Groupware se responsabilizan de la gestión compartida y de la coordinación de las tareas a realizar sobre el documento.

Los servicios de Groupware para la gestión documental acostumbran a utilizarse para organizar aplicaciones autónomas. Su arquitectura es distribuida, mayoritariamente C/S y implementada a través de paquetes estándares.

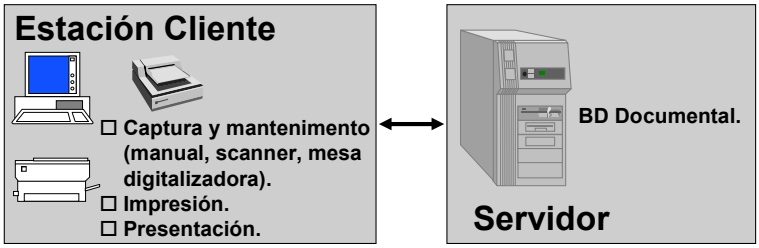


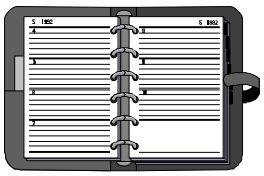
Figura 49. Gestión de un Workflow documental

La relación con las aplicaciones de gestión convencionales suele ser de inserción y consulta, está última siempre a través de propio producto de Groupware.

Por ejemplo, imagine que diseña una aplicación de recepción de albaranes y su empresa tiene implementada una base de datos documental. En un momento de su aplicación deberá escanear e incluir el albarán dentro de esa base documental. En ese momento deberá llamar al servicio de Groupware que lo haga.

Cuando en cualquier momento posterior un usuario de su aplicación necesite visualizar el documento necesitará llamar el servicio que se localice y visualice en pantalla.

4. 2. Sheduling y Calendaring.



La gestión compartida de agendas es otro de los grupos de servicios disponibles en Groupware.

Se permite compartir agendas y calendarios que afectan a más de una persona (reuniones, cursos, etc.). Siempre que se esté autorizado, se puede consultar y anotar en agendas de otras personas.

La generalización del uso de PDA's no ha hecho más que potenciar estos servicios.

A pesar de este bloque de opciones de Groupware no es muy espectacular, permite cosas organizativamente tan potentes como:

- Consultar los compromisos de otras personas antes de asumir compromisos.
- Convocatorias masivas.
- Anotar de forma automática obligaciones de reportar cosas.
- Gestión de las agendas de los directivos por sus secretarias, etc...

No menosprecie los servicios de calendario. Si tiene que organizar una reunión entre personas distribuidas por toda la geografía española, hacer coincidir un

día laborable común entre las fiesta nacionales, de la comunidad autónoma y las locales no es ninguna obviedad.

La gestión directa desde programa puede permitir dar soluciones imaginativas a funciones de una aplicación distribuida.

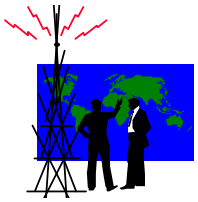
Imagine que está diseñando una aplicación para gestionar un servicio de reparaciones de electrodomésticos a domicilio. Tiene que gestionar las agendas de los empleados de reparación que están en la calle de forma que se optimice su tiempo para dar el mejor servicio posible al menor coste.

Puede asociar a cada empleado una agenda y gestionarlas desde programa. Su programa capturará los avisos de avería y aportará los criterios para optimizar las rutas. Las anotaciones resultantes se registrarán sobre la agenda donde cada empleado tendrá, además de los avisos, registradas todo el resto de sus obligaciones diarias.

Y sus empleados podrán consultar su agenda en cualquier momento e través de una PDA en conexión remota.

4. 3. Conferencing.

Engloba un conjunto de servicios dedicados a permitir reuniones con dispersión geográfica.



Obliga necesariamente a utilizar comunicaciones de gran ancho de banda y hardware “top-line”.

No son opciones útiles dentro del diseño de aplicaciones distribuidas.

4. 4. Correo electrónico (Email).

No es necesario recordar que este bloque de servicios permite intercambiar mensajes y archivos entre usuarios. Dentro de los productos de Groupware, incluye servicios de calendario y agenda.

El alcance del correo electrónico es hoy día mundial gracias a Internet.

La utilidad del correo electrónico dentro de las aplicaciones distribuidas es hoy día fundamental. Más adelante, y dentro del bloque de diseño, encontrará un capítulo dedicado al servidor de correo que, como verá, hoy día se implementa en muchos de los casos bajo correo electrónico.

El correo electrónico ha cogido tal importancia en el mundo actual que son muchos los autores que defienden que al correo electrónico, se le debe llamar simplemente correo y que al correo clásico hay que ponerle el adjetivo de “en papel”.

4. 5. Workflow.

Workflow es, además de un tipo de servicio de los programas de Groupware, una forma de integrar aplicaciones distribuidas por C/S. Déjeme que por eso le dedique un capítulo propio.

Workflow

1. Introducción.

Workflow es un bloque de servicios de Groupware cuya estructura explicaré en este capítulo.

Dentro de esa estructura, los puntos activos pueden ser ocupados por elementos C/S, tanto en Sistemas Operativos como en Internet. De ahí la fuerte interacción entre Workflow y aplicaciones distribuidas.

Desde los orígenes con numerosa y variada oferta, el Workflow se ha integrado como un componente más de ERP's como SAP o se han estabilizado unos pocos productos muy potentes que integran tanto sistemas operativos como en Internet diferenciando los componentes de software de los que interaccionan personas.

2. ¿Qué es Workflow?

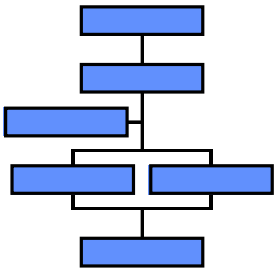


Figura 50. Representación de un Workflow de proceso

Es una tecnología de arquitectura distribuida que permite integrar y dirigir automáticamente operaciones (trabajo y eventos) de un programa o usuario al siguiente.

Según el *Workflow Management Coallition*:

> Workflow es la automatización, total o parcial, de un proceso de negocio en el cual documentos, información o tareas, pasan de un participante a otro para llevar a cabo una acción de acuerdo con un conjunto de reglas de negocio.

La imagen del paradigma es la un barco que navega por un río llevado el flujo de puerto en puerto, donde en cada parada se va incorporando valor añadido.

El trabajo original debe mezclarse con otros, transformarse y dirigirse hacia otro punto del Workflow.

Workflow define las operaciones que deben de hacerse a lo largo del camino y cuando ocurre una excepción (evento) se ejecutan algunas de esas operaciones.

Workflow supone colaboración entre más de una persona, grupo, departamento y hasta compañías.

Es especialmente aplicable a organizaciones medias y grandes que mueven grandes documentos y tienen complejos flujos en sus **procesos de negocio**.

Una forma habitual de diseñar un proceso de negocio en Workflow es integrarlo en un servicio.

3. Elementos del Workflow.

3. 1. Agentes.

Son los elementos activos que aportan valor añadido. Son básicamente personas y programas.

3. 2. Objetos.

Son los elementos pasivos sobre los que trabajan los agentes. Se incluyen dentro de la definición de objetos: documentos, formatos, eventos, mensajes, etc.

3. 3. Rutas.

Definen los caminos (***path***) por los cuales los objetos han de viajar.

3. 4. Reglas.

Definen las condiciones bajo las cuales los objetos han de viajar y a que agentes deben visitar. Por ejemplo, si en un proceso de venta el valor de la compra es superior a 5.000 Euros, el pedido debe viajar al Dpto. de Créditos para autorización.

3. 5. Rols.

Definen los lugares de trabajo y la gente que los ocupa. Por ejemplo, la autorización de crédito lo cubre el rol de Aprobar Crédito y el rol lo ocupa Juan y, si él no está, Luis.

3. 6. Usuarios.

Son las personas que asumen los rols. Pueden gestionarse, y de hecho se gestionan, dinámicamente.

4. Modelos de Workflow.

Los elementos de Workflow se integran en **modelos** que son los que definen los procesos de negocio.

Los modelos se clasifican por diversos criterios.

4. 1. Clasificación basada en las tres R's.

Rutas, Regles y Rols se conoce con las tres R's. Según se agrupen se definen dos modelos de Workflow:

4.1.1. Workflow orientado al proceso.

La base es la ruta. El trabajo es básicamente secuencial. Es el modelo que tiene más incidencia con diseños distribuidos.

4.1.2. Workflow “ad hoc”.

Orientado a rols. El trabajo es básicamente incremental.

Por ejemplo, hay que construir un documento multimedia que integra texto, fotos y vídeo. Cada tarea se asigna a un especialista y existe un director artístico responsable del documento.

4. 2. Clasificación basada en la complejidad y estructura de las tareas.

4.2.1. Ad hoc.

El proceso está muy poco definido. El punto importante es la coordinación y colaboración entre las personas que participan. Los participantes tienen potestad de modificar el orden de las tareas.

Las herramientas habituales son el correo electrónico y procesadores de texto. Un ejemplo puede ser la elaboración de un presupuesto de venta.

4.2.2. Administrativo.

Ataca procesos repetitivos y previsibles. La mayoría de los procesos son manuales. Los participantes no deciden el orden de las tareas. No acostumbra a haber grandes volúmenes de datos y documentos.

Son herramientas habituales las de ofimática y programas especializados. Un ejemplo puede ser un control de petición y control de gastos de viaje.

4.2.3. Producción

Ataca también procesos repetitivos y previsibles pero con res características diferenciales:

- Hay una implicación de BD heterogéneas de gran volumen e incluso dispersas geográficamente.
- Las tareas son automáticas.
- Hay sistemas expertos.

La frontera entre administración y producción no es, de cualquier forma, muy clara.

Un ejemplo puede ser la gestión de compra de un coche de la que hablaremos a continuación.

4. 3. Clasificación basada en el soporte tecnológico.

Esta clasificación, a mi juicio de poco interés ya que clasifica los modelos de Workflow según el soporte tecnológico que utilizan:

- Basada en correo electrónico.
- Basada en programas de gestión de documentos.
- Basada en programas de proceso.

5. Ejemplos de Workflow de datos: Sistema de gestión de datos (SGD).

Los sistemas denominados como Sistemas de gestión documental (SGD) o sistemas para la Gestión de Contenidos Empresariales (ECM del nombre en inglés) son ejemplos de Workflow de datos en los cuales la admisión, catalogación u organización de los datos se realiza desde la fuente.

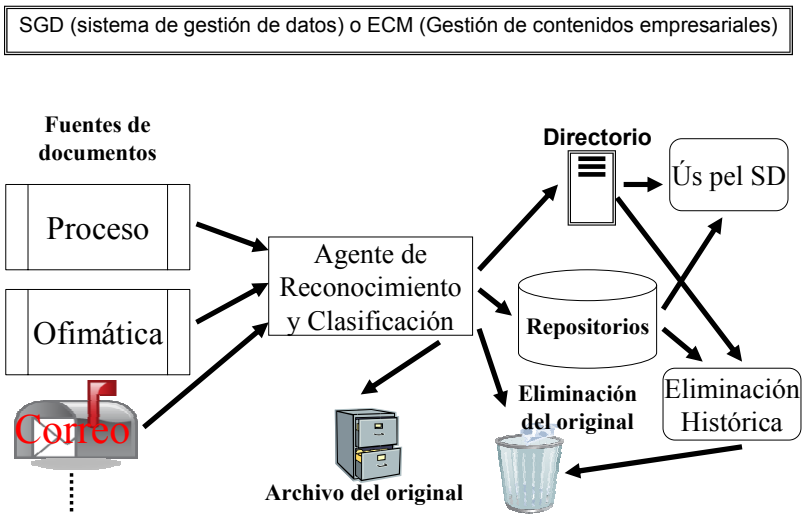


Figura 51. Ejemplo de Workflow de dades.

En la figura de la izquierda se muestra una posible arquitectura para establecer un sistema SGD. Todo el conjunto está regido por el modelo de Workflow establecido. La figura autoexplica claramente su funcionamiento. Observemos que la actitud de la gestión es activa, clasificando “antes” de admitir el documento, archivo, correo, o cualquier otra fuente de información. El SGD cubre todo el ciclo del documento o dato.

Obviamente, la gestión de los correos solo debería incluir los que no son privados. Vamos a entrat en polémica: ¿los correos recibidos en direcciones de trabajo pueden considerarse privados? Las empresas lo tienen muy claro: NO. La ley parece definirse por el sí.,

En la segunda parte de libro comprobaremos como esta arquitectura puede ser implementada fácilmente con los elementos de diseño que aportaremos.

6. Ejemplo de Workflow de proceso.

Un ejemplo de Workflow de proceso se muestra en el siguiente diagrama que puede representar, de forma obviamente simplificada, el proceso de negocio de la venta de un coche.

La petición de la compra de un cliente se registra en un pedido, representado por un objeto documento, que viaja por los diferentes departamentos de la empresa involucrados en el proceso de negocio que van incorporando los resultados de su trabajo en el documento.

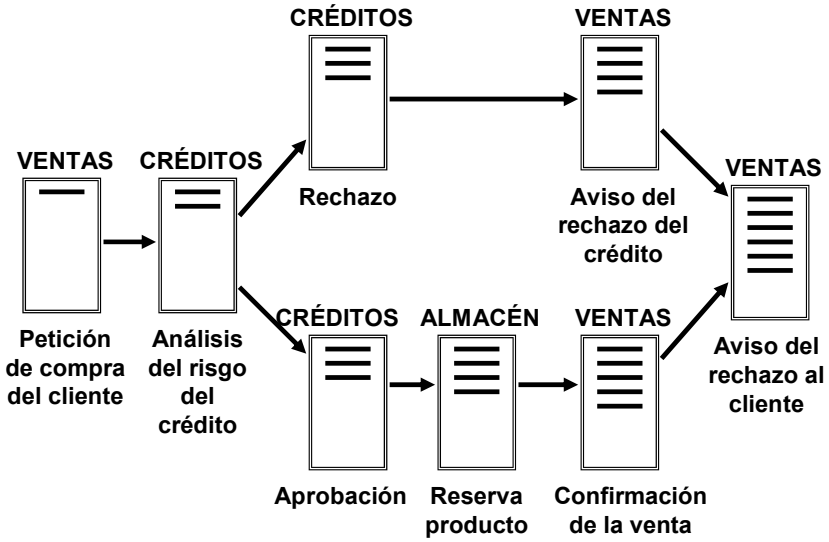


Figura 52. Ejemplo de Workflow de proceso.

Evidentemente dentro de un modelo es habitual la dispersión geográfica. Así, en el ejemplo, los lugares de venta pueden ser delegaciones, el departamento de créditos está en la central y el almacén el la fábrica.

7. Procesos de Negocio y Workflow.

En el ejemplo anterior se ha integrado mediante Workflow un **proceso de negocio**.

> Un **proceso de negocio** es un conjunto de procedimientos (automáticos o manuales) y documentos ligados entre si para asumir un objetivo de negocio de la empresa.

La especificación de las reglas de negocio y los documentos necesarios se modela por **Workflow**, La automatización, total o parcial, se conoce como **Gestión de Workflow**. Si se utiliza una solución basada en un producto especializado, el motor de ese producto para hacer correr el modelo de Workflow se denomina **Gestor de Workflow** (SGW).

8. Componentes de un Gestor de Workflow.

En la figura de la derecha se muestra un esquema de los componentes de un Gestor de Workflow.

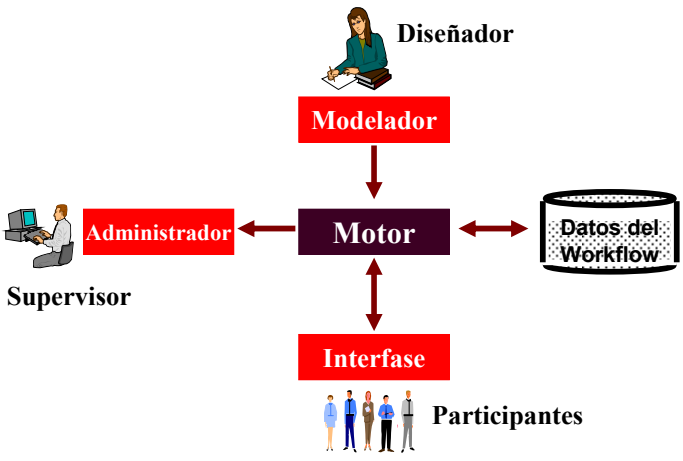


Figura 53. Componentes de un Workflow.

- El Modelador permite al Diseñador definir y mantener el Modelo de Workflow.
- El Administrador permite al Supervisor la instalación, personificación y administración.
- La Interfase permite a los usuarios (Participantes) interactuar con el modelo.
- El Motor interacciona con todos ellos y gestionar la base de datos de soporte del Workflow y otras BD de interacción.

9. Gestión de Workflow.

El modelo y el almacenamiento se localizan en servidores, uno de los cuales puede asumir la función de director.

Hay dos tipos de estación cliente:

- Estaciones activas, donde se realiza la creación y el mantenimiento, la impresión y el seguimiento.
- Estaciones pasivas donde se pueden realizar consultas.

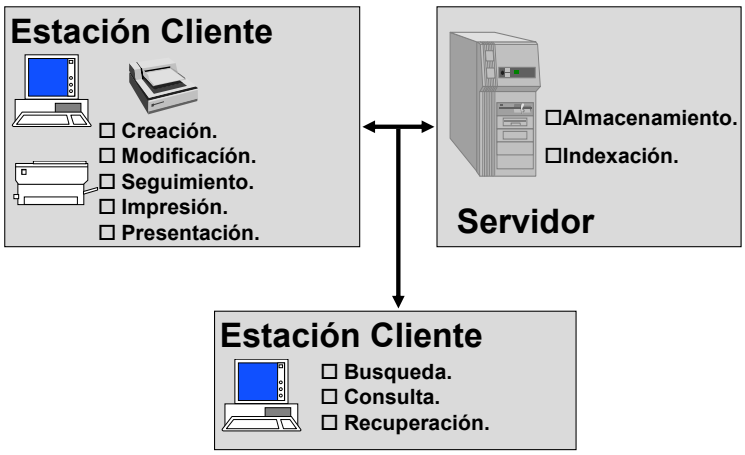


Figura 54. Gestión de un Workflow

10. Workflow y Cliente/Servidor.

Como ya se ha dicho, el muchos casos los agentes son programas. Si se utiliza un paquete de Groupware los programas son procesos del paquete. Pero si además del paquete se necesitan procesos especializados, estos se implementan en programas que, por definición, son elementos (clientes y servidores) de diseño distribuido y, en general, con arquitectura de comunicación C/S, síncrona, asíncrona o desacoplada.

Y en un caso extremo, todos los agentes pueden llegar a ser servicios o agentes desarrollados de forma específica para el proceso de negocio resuelto por el Workflow.

Es decir, y de forma resumida:

- Muchos agentes (elemento Workflow) son agentes (proveedores de servicios C/S).
- Workflow es una forma de integrar aplicaciones distribuidas.

Modelos Distribuidos por Cliente/Servidor.

1. Introducción.

Como en toda rama de la ingeniería en el diseño de aplicaciones C/S es bueno trabajar con modelos. Sin embargo, definir modelos en el mundo C/S, es como mínimo, complicado.

Particularmente considero que definir modelos es un tema fundamental. Por ello a continuación repaso las propuestas de modelos más conocidas, desde las iniciales a las actuales.

Como verá, el tema es realmente confuso y las preguntas salen espontáneamente: ¿Qué diferencias hay entre ellos? ¿Y que relaciones? ¿Es posible definir una relación entre modelos y diseños?

¿A qué no lo adivina? ¡Yo también voy a proponerle un modelo distribuido por una arquitectura cliente/servidor! Mi propuesta está basada en la historia y en mi experiencia. Para convencerle que es la mejor (¡naturalmente!) voy a utilizar tres argumentos:

- Puedo establecer una relación directa entre mi modelo y el diseño.
- Puedo establecer una equivalencia entre todas las propuestas históricas y mi propuesta.
- Se puede atacar con arquitectura de servicios.

2. Lógicas de presentación, datos y proceso.

Antes de entrar a fondo en las diferentes clasificaciones de los modelos cliente/servidor conviene reparar en que cualquier programa puede ser estructurado en tres lógicas: lógica de presentación, lógica de datos y lógica de presentación.

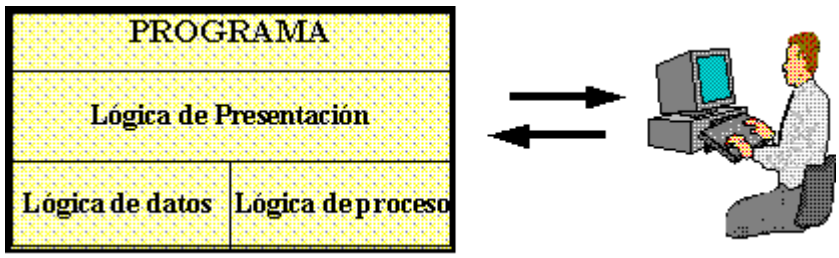


Figura 55. Lógicas de un programa

2. 1. Lógica de presentación.

Se encarga de interaccionar con el usuario. Se implementa integrada en interfícies gráficas con técnicas GUI.

2. 2. Lógica de datos.

Se encarga de resolver la gestión de los datos. Hoy día la base de datos se ataca prácticamente siempre vía SQL, y muy normalmente con ODBC, JDBC o ADO.

2. 3. Lógica de proceso.

Se encarga de implementar la funcionalidad de la aplicación.

Las tres lógicas habrían desarrollarse y de trabajar de forma independiente dialogando entre si de forma transaccional intercambiando mensajes. Que ello sea así depende de la calidad del programa y de la voluntad de conseguirlo.

3. Modelos del Garther Group.

3. 1. Modelo histórico.

En esta figura tiene Vd. un clásico. Nada menos que el primera propuesta (o al menos la primera aceptada por "todo el mundo") que puso orden en la modelación de los sistemas C/S.

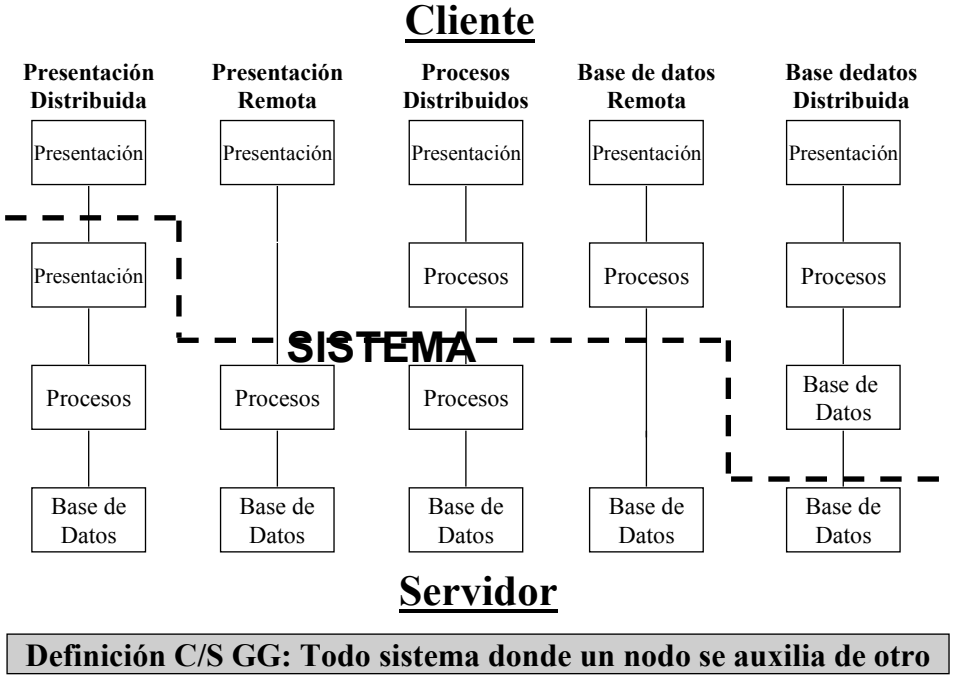


Figura 56. Modelo histórico del Garther Group

Vista ahora le puede parecer simple y arcaica pero le aseguro que durante años la mayoría de artículos sobre C/S la incluían como dogma de fe.

Garther Group estableció como definición de un sistema C/S aquel en el cual los procesos de un nodo se apoyan en los procesos de otro nodo para realizar su trabajo.

Como ve, puro proceso distribuido.

Para clasificar los modelos, la gente del Garther Group partió, muy acertadamente como el tiempo confirmó, de que en toda aplicación, distribuida o no, hay tres partes bien diferenciadas: presentación, datos y proceso. Que estén más o menos diferenciadas depende sólo de la calidad del diseño.

La clasificación se basó en separar las tres lógicas y clasificar todos los casos resultantes de localizarlas entre la máquina cliente y la máquina servidora (recuerde que en esa época no existía el concepto de Middleware). Se incluía incluso una línea a trazos para separar los dos ambientes.

Con este criterio el Garther Group estableció los siguientes modelos:

3.1.1. Presentación distribuida.

Los datos y los procesos se sitúan en el servidor. La presentación se distribuye en dos partes: una parte en el cliente y otra parte en el servidor.

El modelo estaba pensado para aplicaciones transaccionales ya existentes en HOST, que conservando su presentación original (en formato y operativa) se apoyaban en un programa en la parte servidora para mejorar la presentación.

La práctica demostró en seguida que este camino no conducía a ninguna parte ya que el gasto no justificaba el mero cambio estético conseguido debido a que el flujo de diálogo con el usuario no cambiaba.

Pero la historia siguió. Y ahora este modelo "ha resucitado" espectacularmente. **Esta es, ni más ni menos, la forma de trabajo de una WEB**. ¡Le aseguro que los ponentes de este modelo, a finales de los 80's. no debieron estar pensando en una WEB cuando lo plantearon!

3.1.2. Presentación remota.

Los datos y los procesos se sitúan en el servidor. La presentación se hace toda en el cliente. Este modelo se ha utilizado tradicionalmente solo para el maquillaje de aplicaciones (del que se hablará más tarde) sustituyendo el dialogo de los programas realizados en filosofía de monitor de transacciones (CICS, AS/400) por una presentación GUI.

Es la forma habitual de utilizar las aplicaciones C/S avanzadas basadas en sistema operativo.

3.1.3. Proceso Distribuido.

La presentación se realiza en el cliente, los procesos se reparten entre la máquina servidora y la máquina cliente. La base de datos está en la máquina servidora.

Es verdadero proceso distribuido. Si agrupamos el proceso cliente y el componente de presentación en un "programa cliente" y encapsulamos el proceso servidor en servidores, tendremos un sistema distribuido Cliente/Servidor.

3.1.4. Modelo de Base de Datos Remota.

Todo el proceso y la presentación se realizan en el cliente y solo se accede a la máquina servidora para obtener los datos.

Es el modelo más utilizado, con mucho, en la historia de C/S. La mayoría de los programas que se dicen C/S son solo eso, un programa convencional que accede vía ODBC-SQL a un servidor de datos, local o remoto.

3.1.5. Modelo de Base de Datos Distribuida.

Los procesos y la presentación de realizan en la máquina cliente y los datos se reparten entre la máquina servidora y la cliente.

Este modelo y el anterior no difieren en la práctica más que en la administración. Garther Group los separó originalmente debido a que cuando se propuso este modelo se estaba pensando en un HOST como máquina servidora. Incluso en el modelo original en la barrera entre los dos entornos cliente y servidor se dibujaba un teléfono en clara alusión a que el HOST estaba situado “remoto” en relación a los clientes.

Siempre que hablemos, claro está, distribución horizontal (por ejemplo, productos en una BD y clientes en otra) y no de distribución horizontal (los clientes de Girona en Girona y los de Barcelona en Barcelona). Este último caso es habitualmente difícil de gestionar.

A mi juicio, esta clasificación, que supuso un paso fundamental en la evolución de los sistemas distribuidos C/S, nació con la falta de un sexto modelo combinación del proceso distribuido y de datos distribuidos entre maquina servidora y cliente. Este último caso es en el que se situaron rápidamente muchos sistemas distribuidos.

La aparición y utilización masiva de redes locales, que permitieron situar datos y servidores fuera del HOST, y el desarrollo del Middleware, que permitió dejar de hablar de máquinas servidoras y máquinas clientes, desfasó paulatinamente estos modelos de la realidad evolutiva Cliente/Servidor. Pero C/S debe mucho a esta clasificación.

3. 2. Modelo evolucionado.

Como ya se ha dicho al final del apartado anterior, los estilos clásicos de proceso corporativo propuestos inicialmente por Garther Group han evolucionado hacia una arquitectura lógica estratificada en niveles, independizada de la plataforma por el Middleware y distribuida en último lugar según objetos distribuidos.

Ello llevó a que el modelo clásico inicial quedó desfasado y a la propuesta por parte de Garther de un nuevo modelo.

Este nuevo modelo refleja la necesidad de preocuparse en primer lugar por las capas lógicas del proceso C/S y después tomar las decisiones que afectan a la parte física.

Se estructura en las cuatro modelos siguientes.

3.2.1. Modelo tradicional de paso de datos.

Se corresponde con el modelo tradicional de proceso de datos. Un único programa integra todas las funciones de la aplicación y la presentación, tomando los datos directamente de un sistema gestor de bases de datos (SGBD).

Los terminales son no inteligentes.

Se está pensando no solo en la filosofía de grandes ordenadores sino también en aplicaciones sobre máquinas intermedias como AS/400 o UNIX.

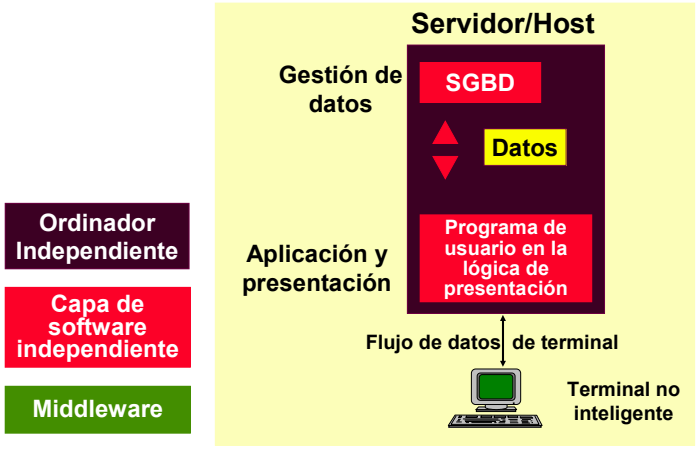


Figura 57. Modelo Tradicional del Garther Group

3.2.2. Modelo C/S biestratificado.

Es un modelo en el cual es la máquina cliente reside la lógica de proceso y la lógica de presentación y la base de datos de encuentra en una máquina servidora.

De hecho, corresponde a modelos de aplicaciones de acceso a BD compartidas en que todo se hace en el cliente. Es decir, la mayoría de las aplicaciones clásicas C/S de consultas y/o mantenimiento de datos, pero siempre con diseño convencional.

El único servidor lógico real es el que proporciona el Middleware y que encapsula el acceso a la base de datos. Son habituales accesos a través del estándar ODBC-SQL.

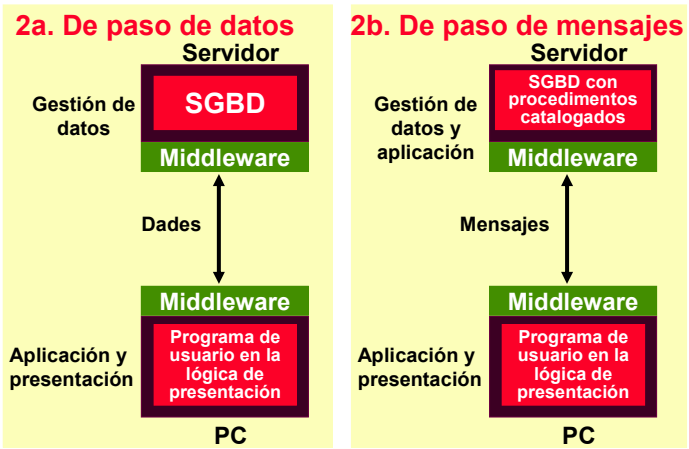


Figura 58. Modelo C/S biestratificado del Garther Group

Hay dos submodelos:

3.2.2.A. De paso de datos.

Esta versión del modelo es la convencional. Si imaginamos que estamos ante un motor SQL, el programa se limita a realizar

llamadas a su través al servidor de la parte de la BD. Los datos viajan del servidor de datos al cliente en su totalidad.

Cuando el tráfico de datos es importante, los tiempos de respuesta pueden ser lentos, especialmente si el número de clientes simultáneos es alto.

3.2.2.B. De paso de mensajes.

Para solventar este problema, y siempre que las necesidades de la aplicación lo requiera, se puede usar este segundo submodelo de paso de mensajes.

La idea es hacer una llamada a la base de datos pidiendo una información resumida o sumarizada.

Para conseguirlo, existen dos recursos:

- Encapsular la función de resumen o sumarización en un servidor de datos instalado en la máquina servidora.
- Utilizar procedimientos catalogados, es decir, fragmentos de programa que se incluyen en la BD de datos y que son muy eficientes. De este recurso, fundamental en aplicaciones C/S con gran tráfico de datos, se hablará ampliamente más adelante.

En ambos casos, el dialogo Cliente/Servidor es un mensaje de petición con los parámetros de la sumarización o resumen y un mensaje de respuesta con la sumarización o resumen requerido.

3.2.3. Modelo C/S triestratificado.

La lógica de presentación se encuentra en la máquina cliente, los procesos se reparten entre la máquina cliente y la servidora y los datos están en un servidor de datos.

Las funciones de proceso situadas en máquinas servidoras se encapsulan en servidores, programas de usuario en el modelo.

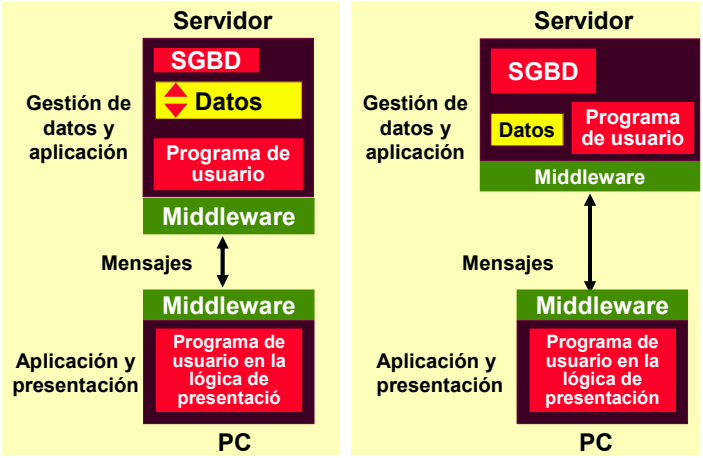


Figura 59. Modelo C/S triestratificado del Garther Group

Existen también dos submodelos:

- El de izquierda en el cual los datos se trabajan siempre encapsulados en servidores.

· Y el de la derecha en el cual los datos se trabajan a través del Middleware.

3.2.4. Modelo de objetos distribuidos.

Es un modelo basado en que todos los procesos y los datos se encapsulan en objetos distribuidos programados con técnicas de OO.

El componente de presentación, un objeto distribuido más, utiliza el resto de objetos.

La distribución de objetos por el sistema se realiza a través de las opciones que proporciona el modelo de objetos distribuidos, J2EE, CORBA o .NET/DCOM.

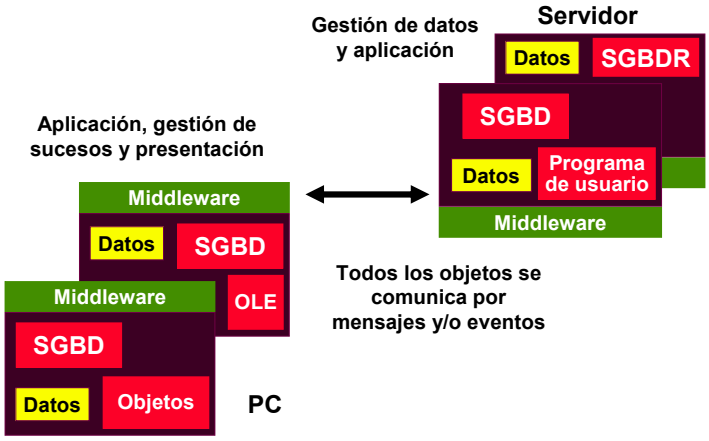


Figura 60. Modelo C/S de objetos distribuidos del Garther Group

Este modelo corresponde más a una técnica de implementación que a un modelo de desarrollo.

Una arquitectura de Servicios WEB, tiene también este aspecto.

Estos nuevos modelos del Garther Group están mucho más acorde con la situación real, pero a mi juicio presentan dos deficiencias:

- Sigue hablándose de máquinas físicas Servidor y Host.
- Separa el Middleware que fabrica el usuario del que se compra la construido,

En cualquier caso, a pesar de ser un modelo más actualizado, continúo pensando que sigue sin ser una buena clasificación desde el punto de vista del diseñador.

4. Topologías de los sistemas distribuidos C/S según la distribución de los componentes básicos.

Hemos visto anteriormente que con la aparición del Middleware es posible reducir los cuatro componentes básicos del sistema a sólo tres: cliente, Middleware (que incluye sistema y transportista) y servidor.

Existe una clasificación de los sistemas C/S en función de donde se localiza el Middleware. Es la clasificación en "**topologías**" que se desarrolla a continuación.

4. 1. Usuarios Nómadas.

Se incluyen en esta clasificación sistemas en los que los tres componentes se instalan en la misma máquina.

Nótese que ello lleva implícito que es ordenador puede trabajar con total autonomía.

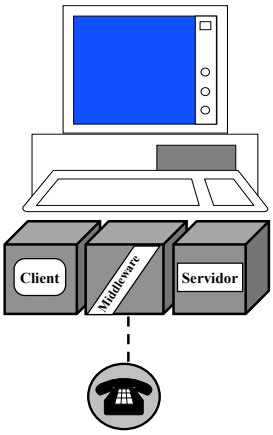


Figura 61. C/S nómada.

Aislado es una terminología que hace referencia a usuarios desconectados habitualmente de la red y que solo se conectan a ella en momentos puntuales.

El término **nómada** hace referencia a usuarios que se desplazan físicamente (vendedores por ejemplo) y que utilizan un portátil.

En este caso es muy frecuente que se conecten de tanto en tanto a las sedes centrales o departamentales para recibir y/o enviar información.

Veamos un ejemplo. Imagine que Vd. tiene una compañía con una red de tiendas. Y que estas tiendas tienen volúmenes muy diferentes. Podrá diseñar una sola aplicación C/S para toda su red de ventas. En las grandes instalará una red y distribuirá en ella los clientes, servidores y el Middleware; en las pequeñas un sólo PC instalará los tres componentes en un único PC. Si la tienda crece no tendrá que cambiar aplicación ni volver a formar a los empleados ni cambiar los métodos de trabajo. Bastará instalar una red y distribuir los tres componentes.

Debido a la limitación de disponibilidad de la red global, las aplicaciones de este tipo de usuarios disponen de procesos de replicación para suplir en local los servicios no disponibles por la desconexión de esa red global y de interfase para almacenar y trasmitir posteriormente los resultados. La información de la red global para estos usuarios es también por interfase.

4. 2. Red homogénea.

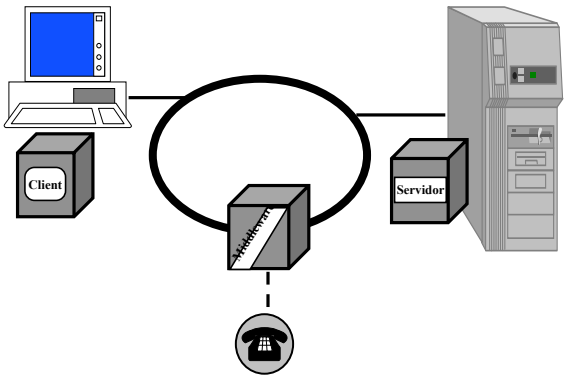


Figura 62. C/S sobre una topología de red homogénea

Son plataformas en las cuales hay total interoperatibilidad, que recuerde que no quiere decir necesariamente igualdad de sistemas operativos, redes y demás componentes de la infraestructura.

Los servicios están disponibles todo el tiempo, hay homogeneidad en el Middleware y la red es

local o remota, pero proporciona la suficiente velocidad de transmisión.

Las comunicaciones remotas se tratan por interfase aprovechando opciones de interconexión.

En esta situación el Middleware no tiene más limitación de localización que los criterios de administración del sistema.

Es una situación ideal para un diseñador ya que no hay puntos de heterogeneidad.

4. 3. Red heterogénea.

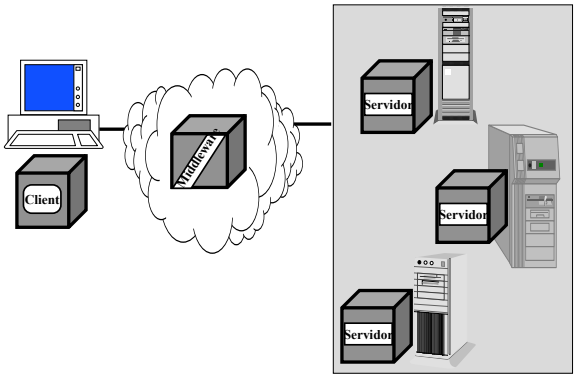


Figura 63. Modelo C/S sobre topología de red heterogénea

Es una topología de red donde no hay interoperabilidad entre todos los nodos.

Ello no equivale necesariamente a comunicaciones remotas.

Ya hemos visto anteriormente que los puntos de heterogeneidad pueden ser debidos a otras causas.

De hecho, este tipo de heterogeneidad puede ser de dos tipos:

4.3.1. Locales.

La heterogeneidad suele venir por la presencia de sistemas operativos diferentes que crean puntos de heterogeneidad en el Middleware.

4.3.2. Remotas.

La heterogeneidad suele venir por la presencia de Middleware heterogéneo y, lo más frecuente, por la presencia de conexiones lentas.

Una topología de red heterogénea exige al diseñador un análisis exhaustivo que permita la detección y enumeración de todos los puntos de heterogeneidad para no llevarse después sorpresas no deseadas.

Si está delante de una red heterogénea estudie y valore concienzudamente el coste de eliminar el máximo puntos de heterogeneidad. Si hace números para invertir en plataforma en lugar de en programas, seguro que en la inmensa mayoría de los casos ahorra dinero, problemas y adaptabilidad.

4. 4. Red global.

¿Soñamos? Érase una vez un mundo donde las comunicaciones remotas y locales funcionaban en todo el planeta a la misma velocidad (la de una red local por supuesto).

Todos los productos Middleware de una misma línea respondían al mismo estándar. Y además se adaptan a él al 100%.

Todos los elementos de hardware eran intercambiables...

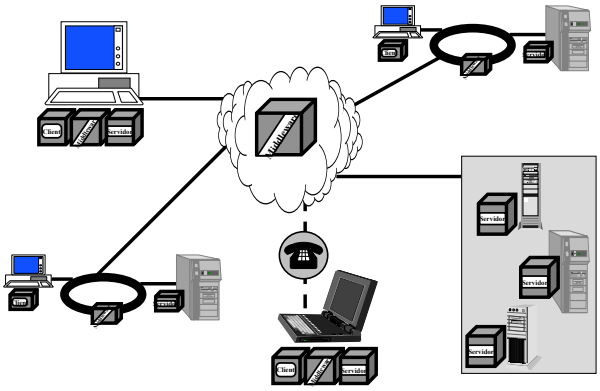


Figura 64. Modelo C/S sobre una red global

Bienvenidos a la red global. Por cierto si Vd. llega a verla, la podrá tratar como una red homogénea a nivel planetario.

Desgraciadamente nos está sonando el despertador... las redes globales actuales son una mezcla de homogeneidad y heterogeneidad.

Como es fácil observar, esta clasificación de los sistemas C/S por topologías puede tener cierto interés para administradores del sistema pero poco para diseñadores. Excepto en un caso, determinar puntos de heterogeneidad, asunto que como sabemos es fundamental en el diseño C/S.

Aun y así, creo que no es la clasificación que interesa a un diseñador de sistemas distribuidos.

5. Modelos de aplicación.

La propuesta de clasificación de los sistemas distribuidos por el modelo de la aplicación está basada en la distribución de la funcionalidad, y aunque la propuesta ha tenido a lo largo de los años diferentes formalizaciones, puede resumirse en el siguiente cuadro:

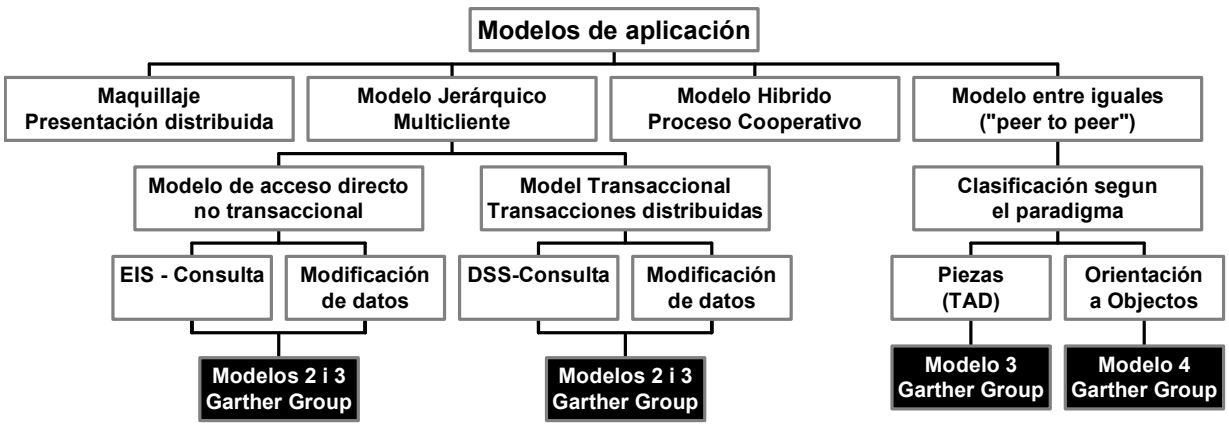


Figura 65. Modelos de aplicación

He colocado en la figura inicial de este apartado la relación, que ha mi juicio, existe entre la clasificación en modelos de aplicación y los modelos del Garther Group.

Comentemos cada uno de los apartados de esta clasificación.

5. 1. Aplicación de maquillaje o de presentación distribuida.

Su característica básica es que en los programas del HOST se ha separado la presentación a los usuarios de proceso. La presentación se hace con un programa GUI que respeta el dialogo transaccional del programa original. Es decir, la parte cliente sólo tiene la lógica de presentación.

Sólo es aplicable de forma operativa a aquellos entornos de HOST de programación transaccional como Mainframe, AS/400, etc.

Hay una máquina importante tipo Mainframe que soporta los datos corporativos.

Todo el proceso corporativo se hace centralizado y los datos y los resultados se piden y presentan por el componente GUI.

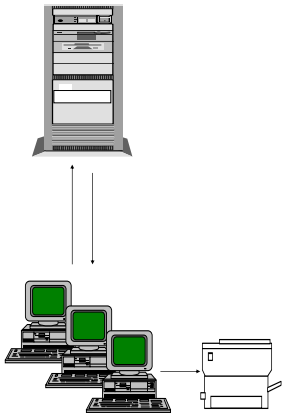


Figura 66. C/S de maquillaje

Puede existir o no una red local ya que los PC's pueden estar conectados directamente a las antiguas líneas de los terminales. Aunque está situación está prácticamente desaparecida.

Las aplicaciones de maquillaje tienen dos objetivos fundamentales:

- Hacer más cómodo y eficiente el trabajo del usuario.
- Ser el primer paso de reingeniería de sistemas que ha de permitir empezar a trabajar en C/S sin cambiar las aplicaciones actuales suministrando el código heredado como servicios.

5. 2. Modelo jerárquico multicliente.

Caracterizado básicamente por la utilización de las aplicaciones cliente/servidor como complemento de las aplicaciones corporativas, básicamente de consulta de resultados.

Hay una máquina importante tipo Mainframe que soporta los datos y los procesos corporativos. Hay redes locales conectadas, local o remotamente, desde los PC's conectados se trabaja a fondo con las aplicaciones corporativas, en emulación o en maquillaje. Los PC's se usan de forma directa sólo para ofimática y, ocasionalmente, para pequeñas aplicaciones de consulta.

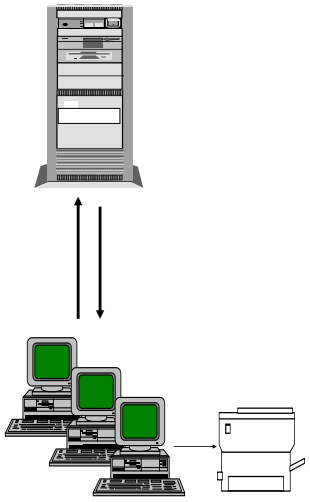


Figura 67. Modelo Jerárquico Multicliente

El proceso corporativo es centralizado.

Es frecuente la existencia de comunicaciones para conectar clientes remotos.

Habitual en soluciones basadas en Internet.

Hay dos submodelos en función del número de usuarios y del número y tipología de las transacciones:

- Modelo simple de acceso directo, no transaccional.
- Modelo transaccional, normalmente basado en Internet.

5. 3. Modelo simple de acceso directo a los datos.

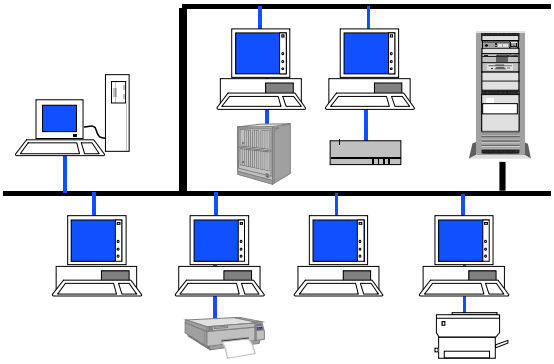


Figura 68. Modelo simple de acceso directo a los datos.

Caracterizado básicamente por la existencia de una red local muy homogénea sobre la cual se procesa una aplicación de diseño muy convencional que utiliza la red como servidor de datos y de impresión.

Es una situación muy habitual en pequeñas y medias empresas.

Hay una base de datos lógica que puede estar formada por varias de físicas homogéneas. El número de usuarios es bajo (¿20-40?). La base de datos se utiliza a través de SQL.

El crecimiento de la red ha podido provocar la instalación de un servidor de datos especializado de gran capacidad de almacenamiento y proceso.

Existen dos tipos muy diferenciados de programas cliente:

- Consultas.
- Gestión de los datos: registro y modificación.

5. 4. Modelo transaccional.

Es un modelo de aplicación caracterizado por la existencia de una gran BD sobre un ordenador de gran capacidad de almacenamiento y proceso. La BD contiene datos corporativos para que puedan ser consultados por un gran número de usuarios.

Lo normal es que la BD contenga datos replicados y sumarizados de la BD principal sobre un Data Warehouse y similares.

Debido al gran número de usuarios y a la potencia de las consultas, los productos de software utilizados son de filosofía transaccional. Se necesita, pues, un gran motor de BD con un monitor transaccional.

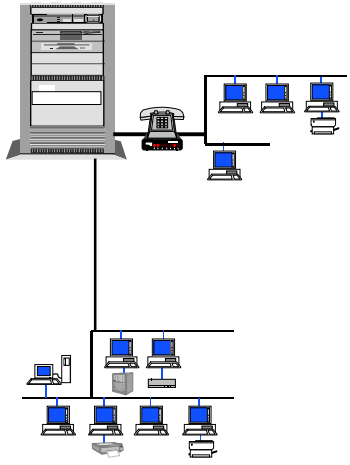


Figura 69. Modelo de aplicación transaccional

El gran número de usuarios y transacciones consultando simultáneamente obliga a asegurar tiempos de respuesta razonables en los momentos punta.

La solución final debe permitir la ampliación cómoda y rápida del número de usuarios.

La conexión del entorno servidor se realiza habitualmente por red local pero son frecuentes las consultas por red remota.

Por su alto coste, tanto de inversión como de mantenimiento, este modelo de aplicación nunca ha tenido mucho éxito fuera de grandes compañías hasta la llegada de Internet. Actualmente las aplicaciones C/S basadas en sistema operativo se está sustituyendo por soluciones de Intranet y Internet y se están creando nuevas soluciones en este modelo basado en Internet.

5. 5. Modelo híbrido: jerárquico multicliente + servidor.

Es un modelo de aplicación caracterizado por la existencia de una gran máquina central tipo HOST y redes departamentales. Las redes departamentales suelen ser de tamaño mediano o grande con potentes servidores.

La conexión del entorno servidor se realiza habitualmente por red local pero son frecuentes las consultas por red remota y las conexiones de los servidores departamentales a la red global suelen ser de gran capacidad. Es habitual también la presencia de impresoras compartidas de gran capacidad de impresión.

En el HOST realizan los procesos corporativos y interdepartamentales y en la red de cada departamento se implementan de forma autónoma las aplicaciones departamentales, no sólo de consultas sino también de gestión.

Los datos específicos de cada departamento están en su servidor departamental y existe en el HOST una BD replicada y sumarizada de los datos corporativos e ínter departamentales.

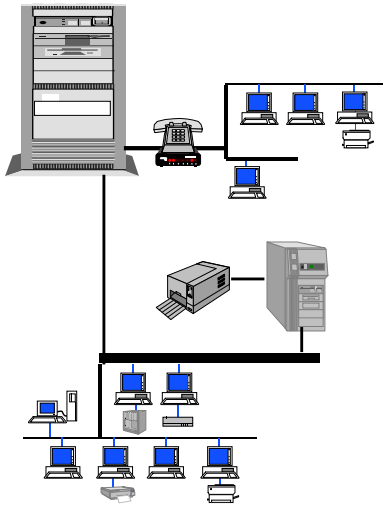


Figura 70. Modelo de aplicación hibrido

La aplicación total es una mezcla de los otros modelos con fuerte presencia de los modelos jerárquicos y de acceso directo a los datos.

5. 6. Modelo entre iguales (peer-to-peer).

Es un modelo de aplicación caracterizado por ejecutarse sobre una plataforma donde no hay una gran máquina que realice un proceso corporativo centralizado. Este proceso corporativo está repartido por un conjunto de servidores de gran capacidad interoperables entre si.

Las aplicaciones suelen ser de diseño muy modular, con muchos servidores. Se necesita personal formado para diseñarlas.

Los datos están en bases de datos distribuidas en dos sentidos:

- BD corporativas y distribuidas.
- BD locales en un servidor.

La conexión de los servidores se realiza habitualmente por red local de gran capacidad de transporte. Hay también conexiones de redes remotas, generalmente para procesos obtención de datos o de interfase.

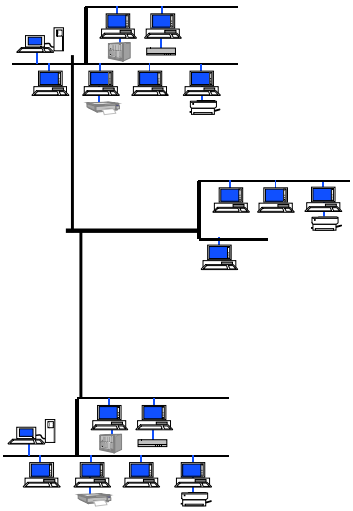


Figura 71. Modelo de aplicación entre iguales.

Se observa claramente que esta clasificación de los sistemas C/S por los modelos de aplicación es de muy dudosa (por no decir nula) utilidad para modelar un diseño ya que se mezclan concepto de plataforma y funcionalidad y además de una forma que no ayuda a clarificar el diseño, uno de los objetivos básicos cuando se establecen modelos.

6. Modelo de Servicios WEB.

Recuerde aquí la arquitectura de Servicios WEB presentada anteriormente que es modelo más de aplicaciones C/S.

El modelo consistiría en utilizar solo este tipo de servicios. Parece, a priori, tan restrictivo que aplicarlo en una realidad tan compleja como un sistema distribuido huele más a teoría que a modelo práctico.

7. Modelos de Diseño.

Personalmente creo que los modelos que hemos visto hasta ahora no me ayudan en mi función de diseñador de aplicaciones distribuidas. Así que le propongo que trabajemos con otra clasificación.

La clasificación que denomino **Modelos de Diseño está basada**, no en donde si no, **en como se agrupan las lógicas de presentación, proceso y datos**.

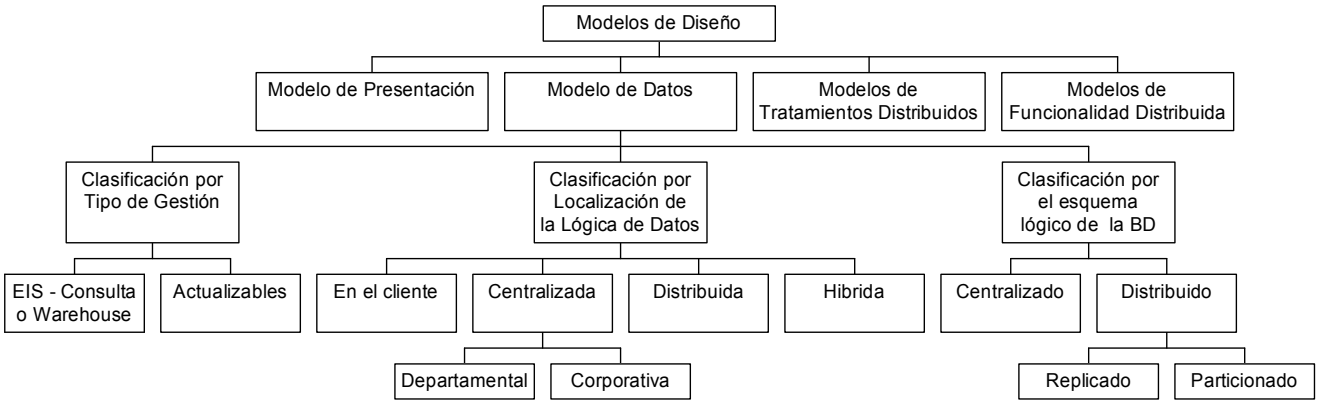


Figura 72. Modelos de diseño.

Además, esta clasificación permite reflejar sin ningún problema una realidad de los sistemas distribuidos: un mismo sistema puede compartir más de un modelo.

Veamos a continuación cada uno de ellos.

7. 1. Modelo de presentación.

De este modelo ya se ha hablado ampliamente en clasificaciones anteriores.

En la parte cliente nada más está la lógica de presentación que trabaja con una arquitectura de distribución con un servidor que proporciona la funcionalidad. Más adelante se hablará de la arquitectura de distribución dentro del capítulo dedicado a la arquitectura de servicios.

7. 2. Modelo de Datos.

Corresponde al modelo de aplicaciones basadas en una BD centralizada. Son modelos en que la lógica de presentación, la lógica de proceso y la mayor parte de la lógica de datos se agrupan en el cliente y sólo una parte de la lógica de datos, normalmente muy pequeña, de sitúa en servidores.

En la mayoría de los casos el cliente sólo utiliza del Middleware los servidores de impresión y los servicios SQL del servidor de datos asociado al motor de la base de datos.

A pesar de ser el modelo clásico, o quizás por que ello, sigue siendo el modelo más utilizado y el que presenta más especializaciones. Y observe que la llegada de Internet ha reforzado esta tendencia.

Existen dos grupos genéricos:

- Los modelos de acceso directo no transaccional, que utilizan motores SQL.
- Los modelos transaccionales, que utilizan para su diseño lógica de transacciones.

A continuación se presentan subclasificaciones de modelos de datos, aplicables tanto a modelos transaccionales como no transaccionales, que son ortogonales entre si. Es decir, que un mismo modelo de datos participa de un submodelo en cada dimensión.

7.2.1. Modelos de datos de consulta o actualizables.

Es una clasificación basada en el tipo de gestión que se realiza con los datos:

7.2.1.A. Modelo de consulta.

Llamado a veces modelo de Data Warehouse (ver clasificación de los modelos de datos). Son aplicaciones donde no se realizan nada más que consultas a la base de datos. Ello implica que la aplicación no ha de preocuparse de la consistencia de los datos.

7.2.1.B. Modelo actualizable.

Los datos son actualizados por la aplicación que debe garantizar la coherencia y consistencia de los datos.

Obviamente, a efectos de diseño la importancia de este grupo radica en que la aplicación debe garantizar la coherencia de los datos, cosa que, como cualquier diseñador sabe, no es ninguna trivialidad.

7.2.2. Modelo centralizado o distribuido.

Es una clasificación basada en el tipo de esquema lógico de la base de datos.

Aunque este tema se tratará en profundidad más adelante, veamos las diferentes especializaciones que tiene este grupo.

7.2.2.A. Modelo centralizado.

Un único esquema lógico. La centralización puede ser corporativa o departamental.

7.2.2.B. Modelo distribuido.

Hay dos especializaciones:

7.2.2.B.a. Distribución vertical.

La información se distribuye por entidades en más de un esquema lógico. Por ejemplo los clientes en la base de datos de comercial y los productos en la base de datos de fábrica.

Hay más de un esquema lógico pero el tratamiento es como el modelo centralizado, aunque el programa tenga más de una base de datos abierta.

7.2.2.B.b. Distribución horizontal.

Hay entidades distribuidas en más de un esquema lógico. Por ejemplo, los clientes de Girona están en un esquema y los de Barcelona en otro. Es una situación normalmente compleja.

La influencia de estos submodelos en el diseño puede resumirse en dos puntos de heterogeneidad que pueden estar o no presentes en cada caso concreto:

- API's de gestión de la base de datos con diferencias semánticas y sintácticas diferentes.
- No disponibilidad de Middleware para tratar los modelos distribuidos horizontalmente.

7.2.3. Remoto o local.

Es una clasificación basada en la velocidad de acceso a los datos, cualidad que normalmente es una consecuencia de la localización remota o local de las máquinas servidoras y cliente y la presencia en el primer caso de comunicaciones "lentas".

La influencia en el diseño es, como ya se imagina, la aparición de puntos de heterogeneidad por un transportista lento.

7.2.4. Clasificación por la localización de la lógica de datos.

Tiene cuatro especializaciones:

7.2.4.A. Lógica de datos en el cliente.

El cliente solo utiliza los servicios de SQL del Middleware. Es la situación más habitual en este subgrupo.

7.2.4.B. Lógica de datos centralizada.

La lógica de datos está toda fuera del cliente, en servidores especializados y/o procedimientos catalogados.

7.2.4.C. Lógica de datos distribuida.

La lógica de datos está distribuida por varios puntos de la organización.

7.2.4.D. Lógica de datos híbrida.

El diseño participa de las especializaciones anteriores.

La lógica de datos centralizada presenta una mayor dificultad de diseño y programación que la residente en el cliente pero en contraposición presenta un mejor rendimiento y favorece la gestión de la coherencia de los datos.

No hay una fórmula mágica para escoger. El diseñador habrá de estudiar detenidamente para situación y escoger en función del rendimiento y la seguridad en la coherencia de los datos que necesite.

7. 3. Modelo de tratamientos distribuidos.

El diseño presenta gran cantidad de servicios especializados en tratamientos concretos. La inmensa mayoría de estos servicios los proporcionan servidores o escasean o no existen los agentes.

Toda la lógica de proceso está en el cliente pero delega muchos tratamientos especializados en los servicios.

La distribución final del proceso está muy compensada entre clientes y servicios.

7. 4. Modelo de funcionalidad distribuida.

Hay servicios que realizan funciones de alto nivel lo que comporta que la funcionalidad y la lógica de la aplicación está distribuida entre clientes y servicios. Coexisten servidores y agentes como proveedores de los servicios.

La distribución de la funcionalidad puede distribuirse entre:

- Clientes **(nivel cliente**).
- Servicios de funcionalidad (**nivel funcional**).
- Servicios de tratamientos **(nivel de tratamientos**) que incorporan lógica de negocio.

Es un modelo de diseño complejo que necesita personal formado. Si no es así, puede llevar a la creación de sistemas muy inestables y peligrosos.

Modelos de Desarrollo de Aplicaciones Distribuidas

1. Introducción.

Desde el punto de vista del modelo de análisis y desarrollo existen dos tipos básicos de aplicaciones distribuidas. La decisión por uno u otro camino se basa en las prestaciones que se esperan de la aplicación.

Así, se habla de aplicaciones de diseño rápido y de diseño avanzado. Las presentamos en este capítulo.

Ambos modelos de desarrollo son aplicables tanto a desarrollo de aplicaciones distribuidas basadas en Sistema Operativo como a las basadas Internet. Por esa razón, todo lo que presentemos en este capítulo es aplicable a ambos entornos.

2. Aplicaciones de Desarrollo Rápido (RAD).

Las aplicaciones RAD corresponden a aplicaciones de sistemas de información en las cuales el **diseño se centra en interfícies gráficas que se gestionan datos de bases de datos relacionales utilizando solo dos niveles de desarrollo**. Si el diseño necesita **otros servicios, se utilizan los ya creados** dentro del entorno distribuido.

Los contenidos de las entidades registradas en la BD son en la mayoría de los casos replicas y sumarizaciones de datos de gestionados por otras aplicaciones de "mayor nivel". Cuando los datos a los que se acceden son los de las entidades originales los usuarios solo tienen, en la mayoría de lo casos, derechos de lectura.

Son aplicaciones básicamente de:

- Consultas por pantalla y listados puntuales. Sólo en algunas ocasiones, a parte de tablas propias de la aplicación, se modifican datos de la base de datos relacional, suelen ser totalizaciones o parametrizaciones y en estos casos los problemas de consistencia suelen ser mínimos.
- Registro de datos distribuidos, como por ejemplo, capturas remotas de pedidos. Los datos capturados son procesados por aplicaciones corporativas o a aplicaciones departamentales de poca complejidad.

En este tipo de aplicaciones:

- **No suelen haber diseño de servicios**. La aplicación se limita a utilizar servicios ya construidos, comprados como Middleware estándar o desarrollados internamente. Es más, la mayoría de servicios son de datos y son suministrados por un gestor de base de datos.
- El **modelo de datos suele ser una precondición** y la aplicación sólo define tablas propias, generalmente para parametrización y administración.
- Suelen ser aplicaciones con pocas o nulas modificaciones de datos. Y si las hay, no suelen ser críticas y no es complicado garantizar la consistencia.

- No **suele haber agentes** ya que ello supondría definición de flujo entre programas.

Con demasiada frecuencia estas aplicaciones se desarrollan sin ningún tipo de metodología de análisis. El programador recibe del usuario una especificación “oral” y con ella empieza a desarrollar el programa “sobre la pantalla”.

El resultado suele ser muy pobre:

- No hay modelo de datos de las tablas propias de la aplicación.
- Las interfícies gráficas no son reutilizables.
- Los mismos cálculos están duplicados y triplicados.
- Interfícies gráficas parecidas tienen operativas muy diferentes.
- No hay ninguna normalización de los mensajes de error y las operativas de recuperación, etc.

Y finalmente cuando la aplicación se enseña al usuario no responde a lo que éste espera. Se entra entonces en un bucle de reprogramación/validación que cuando deja la aplicación mínimamente operable ha consumido mucho más tiempo y tensiones de las inicialmente previstas. Y además los elementos de la aplicación no son reutilizables y todo el conjunto es difícilmente modificable.

Y lo más grave de todo para una empresa, cuando el programador que ha desarrollado la aplicación “desaparece” ya no hay quien analice un problema y realice los cambios y modificaciones que la evolución de la realidad obliga en todas las aplicaciones.

Seguro que las aplicaciones de este tipo que Vd. lector realiza están el bloque de las pocas que se desarrollan correctamente y opina como yo que una metodología de análisis debe ser empleada también aquí.

Este tipo de aplicaciones tiene un nombre a mi juicio bastante desafortunado: ***Desarrollo Rápido de Aplicaciones***, referenciadas frecuentemente con el término ***RAD***.

3. Aplicaciones avanzadas.

Las aplicaciones avanzadas corresponden a aplicaciones o sistemas de información en los cuales el **diseño se centra en los procesos** y la presentación queda en segundo nivel.

No necesariamente la estructura de datos tiene que ser compleja. En muchos casos es parecida a la que se presenta en aplicaciones RAD. Sin embargo, es este modelo de desarrollo el diseño del esquema de datos existe y está trabajado.

La existencia de problemas de datos distribuidos o de Upsizing ya justifica por si sola la utilización del método avanzado.

Suelen ser aplicaciones en la cuales existen otras integraciones cliente además de la GUI.

El sistema de información acostumbra a estar distribuido en dos bloques.

3. 1. El bloque de arquitectura

Donde:

- Se implementa la arquitectura básica de servicios como una parte de la arquitectura del software. Suele incluir pocos, si hay alguno, programas clientes. Y si existen son en su mayoría agentes implementados como clientes background.
- Se encapsula la coherencia y consistencia de los datos.
- Se encapsulan las funciones básicas de proceso.
- Se resuelven los problemas sintácticos y semánticos de la integración de datos.
- Se encapsulan los puntos de heterogeneidad.

3. 2. El bloque de administración, gestión y explotación

Donde se implementan los programas clientes para administrar, gestionar y explotar la aplicación. Suele tener poca presencia de servicios.

Los tres partes de este bloque suelen estar bien diferenciadas:

3.2.1. Subbloque de administración.

Destinado a los administradores del sistema incluye, además de las funciones generales de administración, funciones específicas de:

- Localización de recursos.
- Parametrización.
- Autentificación de usuarios y recursos.

Los programas de este bloque suelen estar dirigidos a usuarios avanzados o administradores por lo que en su integración hay que primar la información y rapidez de acceso frente a la facilidad de navegación. El diseño de la GUI puede asumir también que el usuario tiene conocimientos informáticos.

3.2.2. Subbloque de gestión.

Destinado a los usuarios responsables de la aplicación y orientado a la parametrización del sistema y la creación y mantenimiento de los ficheros básicos. Dentro del los departamentos usuarios suele haber personal especializado en estas funciones.

Básicamente son los programas de la aplicación que permiten parametrizar el comportamiento básico de las entidades o los registros.

En este tipo de parámetros suele haber dos grupos:

3.2.2.A. Parámetros de comportamiento de modelo de negocio.

Por ejemplo, definir si en una organización los albaranes son o no valorados

3.2.2.B. Datos de calificación de registros de tablas a los cuales se quiere limitar los derechos de modificación.

Por ejemplo si un cliente determinado tiene portes pagados o no

3.2.3. Subbloque de explotación.

Destinado a los usuarios finales. Incluye no sólo programas clientes integrados en interfícies gráficas, sobre sistema operativo o Internet, sino que también los hay integrados en cadenas Batch, circuitos de Workflow o cualquiera de las otras formas de integración que veremos en la segunda parte.

Debe limitar el acceso a los datos de calificación de registros.

Si el modelo de negocio lo necesita, la interfície debe definirse para usuarios poco avezados o con rotación alta primando las facilidades de navegación, el filtro de errores y operativas contextuales no válidas de forma que se posibilite un rápido aprendizaje y limiten los errores de operación.

En ocasiones este bloque se desarrolla con método RAD, aunque en este caso, la calidad del diseño RAD suele ser alta diseñando los servicios no reutilizados en una etapa previa.

Es frecuente que los equipos de trabajo en cada bloque sean diferentes ya que el perfil, la formación y experiencia de los profesionales necesarios es muy diferente. Y también es frecuente que durante el desarrollo del bloque de arquitectura se contrate soporte externo de expertos y especialistas.

Es un diseño basado en componentes y que obliga para trabajar bien a utilizar técnicas OO y todavía mejor, paradigma OO de ciclo completo.

Existe un diseño importante de los servicios y su interacción (arquitectura de servicios). Hay que aportar los elementos de administración del sistema que la plataforma de Middleware no proporcione y hay que diseñar los procesos de recuperación y alternativos en caso de caída definitiva de los servicios (análisis de consistencia).

En unos casos el sistema de información distribuido cubre un proceso de negocio completo. En otros es el verdadero proceso corporativo. En el siguiente apartado de este capítulo le presentaré algunos ejemplos de aplicaciones que se han de desarrollar por el modelo avanzado.

Las aplicaciones avanzadas necesitan una formación en los diseñadores y eso hace que “no sean populares” ya que es muy humano ignorar aquello que es difícil. Lo peor que en una organización puede pasar es que un diseñador poco formado o incompetente ataque el diseño de una aplicación distribuida avanzada como de desarrollo rápido. El fracaso es seguro, el trauma fortísimo y según la importancia de la aplicación en el circulo operativo de la empresa, letal.

En proporción son las aplicaciones distribuidas menos frecuentes. Pero las que verdaderamente justifican que Vd. me esté leyendo.

Créame. He vivido experiencias en mis auditorias, directas o indirectas, de sistemas construidos con gran desviación de costes y plazos como consecuencia de este enfoque equivocado, y que al final han quedado muy poco o nada operativas. Y en algún caso, desgraciadamente sin llegar a estarlo nunca.

Sin embargo, son las aplicaciones distribuidas más divertidas de diseñar. Es de las experiencias más gratificantes que un diseñador formado puede vivir. El esfuerzo y las técnicas que hay que aplicar en las primeras etapas es fortísimo, pero después se pueden conseguir grandes objetivos con poquísimo esfuerzo y en **plazos fiables**. Vamos, ¡el paraíso!

4. Ejemplos de aplicaciones avanzadas.

No pretendo en este apartado presentarle un análisis funcional. Ni tan solo una especificación. Me limitaré a dibujarle cuatro trazos de cada aplicación para que se haga una idea de que entiendo por aplicación desarrollada por el método avanzado.

4. 1. Aplicación de ventas de una tienda.

Es un ejemplo de aplicación departamental que cubre un proceso de negocio.

Piense en una plataforma con un servidor y puestos de venta PC's. Además de los datos, hay otros recursos compartidos: impresoras, líneas de comunicaciones con la central, dispositivos de encriptación, servidores de imágenes, procesos de negocio, etc.

Cada puesto de trabajo debe poder funcionar de forma autónoma ya que en caso de caída del servidor central la tienda tiene que seguir vendiendo (¡sino los clientes se nos irán a la competencia!). Ello obliga a replicar parte de los datos en los PC's cliente y a diseñar procesos de recuperación y de funcionamiento alternativo en caso de ese fallo. Y ha integrar el trabajo realizado durante el tiempo de la caída con el resto de los datos obtenidos en trabajo normal.

Debe prever que hacer cuando falle una impresora en un puesto, debe tener un sistema de autentificación de nivel para los puestos de cobro (hay dinero por en medio), debe consolidar la venta hecha mientras el servidor está sin servicio, etc.

Debe enviar los datos de cada jornada a la central. Y en caso de avería telefónica debe ser capaz de guardar más de una jornada. O de enviar los resultados de una jornada de venta por disquete.

Si el número de tiendas es alto, debe poder mantener las tarifas desde la central. Pero no le basta con enviarlas. Ha de tener confirmación de que han llegado y de que el día en que sean activas, el local las tendrá totalmente operativas y disponibles sin posibilidad de error. Observe que en este caso debe hacer convivir como mínimo dos tarifas: la actual y la del cambio previsto. Y que cambiar de una a otra debe ser trivial y sin probabilidad de fallos.

En fin, siga Vd. mismo; seguro que siguiendo esta línea puede completar muchas más situaciones.

4. 2. Aplicación de seguimiento de una empresa de paquetería.

Es un caso de aplicación corporativa en que el sistema de información es totalmente distribuido.

Un gran servidor en algún lugar de la organización centraliza la información de todos los paquetes que circulan por la Tierra bajo control de la empresa.

Servidores territoriales hacen de niveles físicos para acceder desde cualquier punto de la organización.

Cuando un cliente va a una oficina o encarga por teléfono el envío de un paquete, éste se registra en el terminal local y en el servidor central a través del sistema Cliente/Servidor. Al paquete se le adhiere un código de barras.

En cada etapa del traslado se lee la etiqueta. Así es posible conocer en todo momento la situación del paquete y el tiempo estimado de llegada al destino.

Observe que debe diseñarse una aplicación distribuida ya que el punto a punto es inviable por costes (solo se necesita muy conexión cuando pasa un paquete) y fiabilidad. ¡Se imagina que un paquete se para en un puesto de transito hasta que se recupera la conexión con el servidor central!

Y una aplicación Internet puede permitir que cualquier cliente pueda conocer desde su casa la situación de su envío. ¡Imagine el ahorro de coste de personal atendiendo el teléfono! ¡Y la mejora del servicio!

Además, es este caso aplicaciones tan corporativas como facturación y contabilidad son también distribuidas. Cada país tiene sus propias reglas contables y fiscales.

Finalmente, es seguro que si la aplicación no está basada ya totalmente en Internet, se dispondrá de un módulo para que se pueda seguir pos nuestros clientes la situación de sus envíos.

4. 3. Gestión de almacenes.

Imagine una empresa con varios almacenes centrales, otros intermedios de distribución y almacenes finales.

La gestión de productos, condiciones de compra y reglas de reposición se llevan centralizadas pero la gestión del almacén es responsabilidad de cada centro. Hay productos que se pueden comprar directamente a proveedores y otros que hay que pedir siempre al almacén intermedio o central asociado a cada almacén final.

La comunicación entre almacenes y con los proveedores y clientes se quiere llevar electrónicamente con reglas de negocio pactadas.

Y toda la información se quiere tener permanentemente actualizada en la central.

Bienvenido a la aplicación distribuida, que podrá montar basada en Internet, en Sistemas Operativos o mixta, situación muy habitual.

Si decide centralizar los datos irá a un modelo basado en Internet y si decide distribuirlos a un modelo basado en Sistema Operativo. Deberá ponderar las ventajas e inconvenientes de cada modelo en su caso específico.

En ambas situaciones, la comunicación con proveedores puede hacerse o a través de protocolos RPC, plataformas de correo o Servicios WEB.

5. La frontera y el verdadero sentido de RAD.

La frontera entre las aplicaciones que hay que desarrollar RAD o avanzado no es, como en todos los casos parecidos de la ingeniería, una línea clara. Suele haber una zona gris.

Pero, curiosamente, si Vd. tiene formación y experiencia, verá en seguida cuando debe utilizar uno u otro método de desarrollo. Sin embargo declaro mi incapacidad para proponer unas reglas claras para tomar la decisión.

Dos *no reglas* si que le propongo. No elija en método en función de dos parámetros:

- Su desconocimiento de los métodos avanzados o su miedo a utilizarlos. Antídoto: fórmese.
- La formación de su personal actual. Antídoto: subcontrate apoyo externo y/o forme a su personal. Con ello motivará a su gente que es una de las mejores inversiones que puede hacer.

Se puede marcar un criterio operativo. Si la aplicación RAD necesita:

- Crear servicios.
- Definir agentes.
- Modificar el modelo de datos.

Se pueden pasar estos procesos al equipo de aplicaciones avanzadas para que los definan e implementes. Una vez realizado el trabajo, la aplicación RAD podrá usarlos ya como un servicio más.

La palabra rápido del **concepto de RAD debería venir por la facilidad de integrar aplicaciones rápidamente a partir de componentes ya creados** y fiablemente probados.

De esa forma se podría reaccionar rápidamente a necesidades de los usuarios permitiendo, además, que los usuarios se involucren lo antes posible en la definición y validación de la aplicación actuación que, como todos estamos convencidos y hemos repetido muchas veces, es fundamental para el éxito de los proyectos informáticos.

El uso de metodologías ágiles en la línea de XP (Extreme Programing) puede ayudar que las aplicaciones RAD se diseñen correctamente.

6. Modelos de Desarrollo de Aplicación y Modelos de Diseño.

Aunque no sea de un modo unívoco, hay una relación entre los modelos de Diseño y los de Desarrollo.

La relación habitual es:

- Desarrollo Rápido:
  - Aplicaciones de Presentación.
  - Aplicaciones de Gestión de Datos Centralizados.
  - Parte de las aplicaciones de Gestión de Datos Replicados
- Diseño Avanzado:
  - Parte de las aplicaciones de Gestión de Datos Replicados.
  - Aplicaciones de Gestión de Datos Distribuidos.
  - Aplicaciones de Tratamientos Distribuidos.
  - Aplicaciones de Funcionalidad Distribuida.

La relación entre modelos de diseño y de desarrollo puede resumirse en el siguiente cuadro:

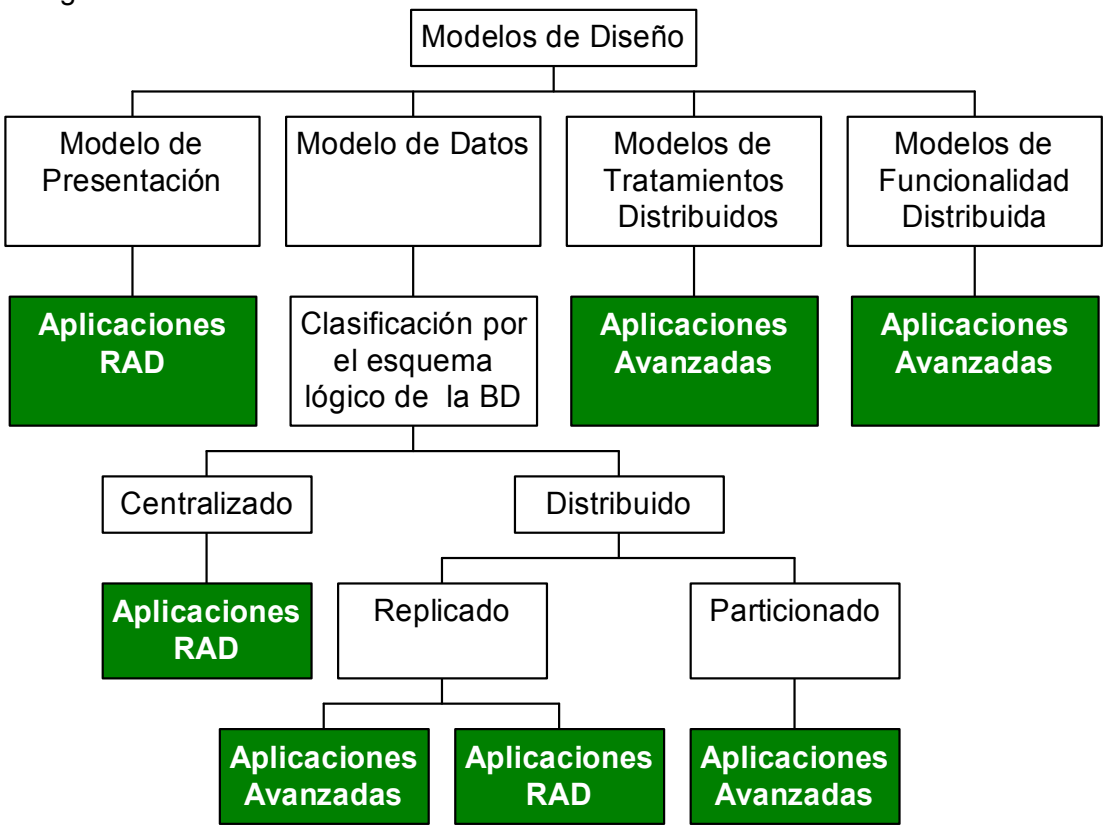


Figura 73. Relación entre modelos de diseño y modelos de desarrollo de aplicación

Introducción al Diseño Distribuido

1. Introducción.

La metodología de diseño de las aplicaciones distribuidas es el objetivo de la segunda parte.

Sin embargo, una vez revisadas las características principales de las arquitecturas distribuidas, ya sean basadas en Sistemas Operativos o Internet, es un buen momento para hacer una reflexión de qué deberá incluir el método de diseño tecnológico para incluir los sistemas distribuidos.

2. Requerimientos y Análisis funcional o Especificación.

Como ya comentamos en el capitulo dedicado a la influencia de una arquitectura distribuida en las construcción de aplicaciones, el hecho de estar dentro de una plataforma distribuida debería tener influencia mínima en las etapas de requerimientos y análisis funcional.

El análisis funcional resolverá las prestaciones necesarias para la aplicación con independencia de si la plataforma es distribuida o no.

3. Posibilidades de diseño por servicios.

Podemos utilizar el diseño por servicios de tres formes:

- Desde la especificación: **Especificación por servicios**:
  - El sistema se contempla como una integración de servicios.
  - Los servicios están ya definidos y especificados.
  - Si alguno no lo está, hay que especificarlo.
  - Los datos son responsabilidad de los servicios.
  - Veremos posibilidades de integración en la composición de servicios (curso de diseño).
- En el diseño: **Diseño por/con servicios**.
  - Hay que identificar y diseñar servicios de datos y procesos.
  - Es una etapa interpuesta en el diseño clásico.
- Solución hibrida: **Servicios como agentes externos**.
  - Es una situación habitual.

Si se ha elegido la primera de ellas, la especificación deja prácticamente resuelta la arquitectura del sistema distribuido, pero el diseñador habrá de estudiar siempre la consistencia y la administración.

En los otros casos se deberá seguir la propuesta que presentamos a continuación.

4. Etapas del diseño Tecnológico.

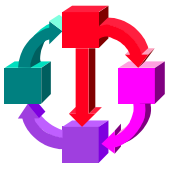
El primer punto donde realmente influye la plataforma distribuida es en el diseño tecnológico.

La presencia de una plataforma distribuida supone, en una imagen visual, la intercalación de una etapa nueva en el diseño tecnológico convencional.

Esta etapa incluye tres fases de diseño.

4. 1. Diseño de la Arquitectura del Sistema Distribuido.

La arquitectura de la aplicación, es lógicamente, distribuida y se construye en el **Diseño de la Arquitectura del Sistema Distribuido**.



Aquí se decide como se fragmenta la funcionalidad en clientes y servicios, que tipo de comunicación se establece entre ellos cual es el modelo de datos y cual es la arquitectura de servicios**.** Los servicios necesitarán en muchos casos de otros servicios. La forma como se llaman unos a otros no puede ser anárquica. Hay modelos preestablecidos entre los que el diseñador deberá escoger. El modelo resultante es lo que se conoce como **Arquitectura de Servicios**.

La base de la estrategia para romper la funcionalidad entre clientes y Servicios será:

4.1.1.A. Fragmentar la funcionalidad.

De hecho, este trabajo ya estará resuelto desde el funcional.

4.1.1.B. Distribuir.

Es el momento importante. Con criterios básicos se analizarán datos, procesos y recursos y se distribuirán entre los clientes y servicios.

Entre estos criterios habrá algunos que se situarán muy claramente en la parte cliente (las presentaciones, por ejemplo) y otros claramente como servicios (la gestión de recursos de hardware compartidos). En medio, una amplia zona de grises que Vd. habrá de analizar, valorar y separar en función de los recursos de Middleware de que disponga, de su instalación y de su aplicación.

Según cargue de funcionalidad a los servicios o a los clientes tendrá aplicaciones ***Fat Server*** o ***Fat Client*** y podrá hablar no de ventajas o inconvenientes de **clientes ligeros**.

Estos conceptos serán tratados en profundidad en la segunda parte del libro.

4.1.1.C. Integrar.

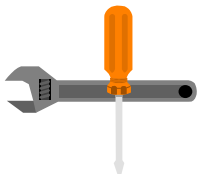
Una vez distribuida la aplicación, habrá de decidir como interaccionan clientes y servidores y agentes integrándolos y localizándolos, además, en la plataforma de sistema.

4.1.1.D. Controlar la explotación.

Ya hemos hablado de que la administración no es un tema simple.

4. 2. Diseño de Consistencia.

El **Diseño de Consistencia** añade la resistencia contra fallos y los procesos de recuperación.



Para construir el análisis de consistencia se analiza cada servicio y se estudia como puede caer, que criticidad tiene la caída, que hay que hacer mientras no se recupera el servicio, como hay que reaccionar cuando se recupera y como hay que reconstruir e integrar lo ocurrido durante la caída con el resto de los datos.

En resumen, el análisis de consistencia establece los procesos de resistencia contra errores y caídas, se establecen procesos alternativos de trabajo y los procesos automáticos de recuperación y reconstrucción.

4. 3. Diseño de Administración del Sistema Distribuido.

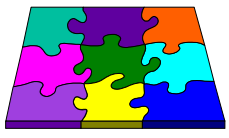
La administración del sistema distribuido es, como se verá más adelante, uno de los puntos débiles de los sistemas distribuidos.

El Middleware, a través de los servicios del DSM, proporciona recursos para la administración del sistema distribuido. Pero en muchas ocasiones, estos servicios no cubren todas las necesidades. El **Diseño de Administración** decide las funciones necesarias y añade y completa los servicios para conseguirlas, ayudando al administrador del sistema a hacer su trabajo.

Recuerde que este es uno de los puntos débiles. Y que administrar es un trabajo de cada día. No menosprecie la importancia de esta etapa para el éxito de su proyecto.

5. Diseño por componentes: la metodología puzzle.

Para implementar los programas clientes, agentes y servidores es fundamental que trabaje con componentes. Y que sus programas estén pensados como integración de componentes ya construidos. Solo la reusabilidad le permitirá costes y plazos razonables.



En los capítulos correspondientes del diseño de la segunda parte le voy a proponer **una metodología puzzle** para construir sus programas: los clientes, agentes y los servidores resultantes del diseño de la arquitectura distribuida del sistema se ensamblan a partir de piezas ya fabricadas. Que le falta una de las piezas, se la construye. Y ya puede utilizar el método de utilizar piezas ya fabricadas.

Para construirlas, deberá usar paradigmas orientados a objetos (OO) o de tipos abstractos de datos (TAD) que le garantizan encapsulamiento y reutilización.

Así, el origen de sus piezas será:

- Middleware estándar.
- Middleware comprado a “terceros”.
- De fabricación propia:
  - Generales a la instalación.
  - Reutilizadas de otras aplicaciones y que así pasan a ser genéricas.
  - Fabricadas específicamente para la aplicación que se esta diseñando.

Sin embargo recuerde que reutilizar es como el buen vino: en su justa medida es una delicia, abusar pone enfermo. Habrá de encontrar el equilibrio entre prestaciones y costes. Hablaremos en la segunda parte de este ratio.

6. Diseño de clientes.

El diseño del cliente se hará diferenciando bien claramente dos factores:

- Las tres lógicas:
  - Lógica de presentación.
  - Lógica de datos.
  - Lógica de proceso.
- La forma de integración de la parte cliente.
  - Recuerde puede tener otras formas de integrar el cliente además de la interfície gráfica (batch, Workflow, etc.)

Haga que la comunicación entre las lógicas sea transaccional.

No hace falta decirle que si hay lógica de presentación será siempre gráfica.

Si sigue estos consejos, y trabaja con metodología puzzle, tendrá mucho ganado y su aplicación será mucho más flexible.

Separa las tres lógicas le permitirá un traspaso fácil de servicios entre servidores y/o agentes y clientes. Imagine que ha tomado la decisión en el diseño de la arquitectura del sistema de dejar un proceso de lógica de datos en un cliente. Y que más adelante necesita convertirlo en un servicio. Si ha trabajado bien, le bastará coger la pieza que implementa esa lógica, crear un servidor y añadirle un modelo de comunicación. Sacando del cliente la pieza e incorporando una simple llamada al servidor, conseguirá que la comunicación deje de ser estática (en fase de link) y pase a ser dinámica (en fase de ejecución). Y el servidor ya estará disponible para dar su servicio a otros clientes o servidores de su aplicación. Habrá de añadir, eso si, al análisis de consistencia correspondiente al nuevo servidor.

Observe además que trabajar así es un seguro. Si se ha equivocado al distribuir las funciones entre cliente y servidor, podrá cambiarlo rápidamente y sin que nadie se entere. ¡Su imagen profesional quedará inmaculada!

7. Diseño de servidores.

El diseño de servidores habrá de ser:

- Especializado a la función asignada.
- Encapsulado. Ha de ser construido como una pieza por si sólo.

- Hay que decidir sensatamente un modo de comunicación con los clientes y servidores que lo utilicen.
- Ha de tener una gestión controlada de los errores.
- La forma de trabajo será:
  - Arrancado siempre.
  - Arrancado a petición.
- La forma de arrancarlo:
  - Dinámica por otros programas.
  - Estática por comandos de sistema.

8. Diseño de agentes.

El diseño de agentes habrá de ser:

- Especializado a la función asignada aunque sea de alto nivel.
- Encapsulado. Ha de ser construido como una pieza por si sólo.
- Ha de tener una gestión controlada de los errores y proveer la forma de avisarlos ya que su ejecución es desatendida.
- La forma de trabajo será:
  - Arrancado siempre, situación habitualísima
  - Arrancado a petición, situación excepcional.

La Arquitectura de Datos

1. Introducción.

La importancia de la arquitectura de datos en el diseño de muchos sistemas distribuidos es básica. Y además uno de sus puntos más conflictivos y que necesita un diseño más metódico.

La falta de ese diseño detallado y cuidadoso ha sido terminante en muchos de los fracasos ya que ha conducido a situaciones en que la inconsistencia de datos ha sido la situación habitual.

Personalmente opino que es hasta tal punto importante diseñar bien la arquitectura de datos en un entorno de este tipo, que más adelante le recomendaré que la analice como primera etapa de su diseño de la arquitectura distribuida.

En este capítulo le presento las nociones básicas necesarias para poder seguir introduciendo los temas de datos en esta primera parte del libro.

De entrada, tenga presente que:

- En un modelo de de datos distribuido hay datos que no están en bases de datos: ficheros secuenciales, XML o no, parametrizaciones, imágenes, multimedia, interfases, etc..
- Las bases de datos son:
  - Relacionales en la inmensa mayoría de los casos (por no decir todos).

- Accedidas por SQL
- Es creciente el uso de:
  - Procedimientos catalogados.
  - SQL como “programación” de los servicios de datos.
- Presentan dos niveles: Lógico y Físico.

- En ocasiones se usa XML como soporte:
  - Temporal.
  - Permanente para evitar administrar un motor de base de datos.

2. Arquitectura de Datos en un Sistema Distribuido.

Entenderemos como Arquitectura de Datos de un Sistema Distribuido la organización física y lógica de los datos sobre la plataforma distribuida.

La Arquitectura de Datos del Sistema Distribuido se corresponde al conjunto de modelos con que se organizan los datos.

Hablar de este tema no es necesariamente hablar de complejidad. Una Arquitectura de Datos válida para un sistema distribuido puede ser centralizada.

Hoy día no se concibe un sistema de datos sin la presencia de una Base de Datos Relacional atacada por SQL. Y en la práctica, los modelos de bases de datos OO no se han impuesto comercialmente y las extensiones del modelo relacional para incluir generalización y especialización son peligrosas por su poca estandarización.

Además la presencia de Microsoft SQL Server es muy importante y se está corriendo el peligro de, para aprovechar mejor sus posibilidades específicas, crear aplicaciones no transportables a otro motor.

Desde el punto de vista del diseño, una solución válida es trabajar con el Modelo de Objetos del paradigma OO que aporta la herencia como una vía estándar de generalización y especialización. Y a la hora de implementar traspasar el modelo de objetos al relacional con las conocidas reglas de equivalencia que encontrará más adelante.

La arquitectura de datos de un sistema distribuido tiene dos niveles el lógico y el físico. Ha habido muchas propuestas para asimilar ambos niveles. Personalmente creo que teníamos la solución delante de los ojos.

En la figura se muestra el esquema ANSI/SPARC de tres niveles de una base de datos.

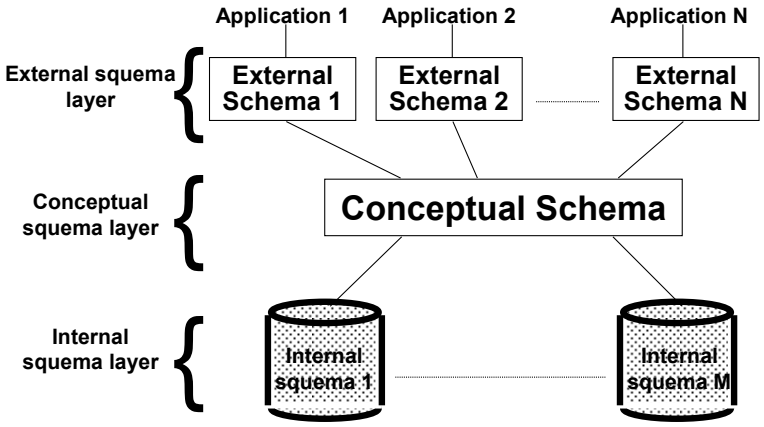


Figura 74. Arquitectura ANSI/SPARC de tres esquemas

Como Vd. ya conoce, la idea básica es que la construcción de una base de datos debe estructurarse en tres niveles:

1. El **Esquema Externo**. Es la visión de la aplicación. Cada aplicación tiene su propio esquema conceptual que es una abstracción del Esquema Conceptual general. Permite aislar la aplicación de los cambios evolutivos de la BD que no la afectan.
2. El **Esquema Conceptual**. Define la información de forma genérica, desde la perspectiva del interés general.
3. El **Esquema Interno**. Implementa el modelo conceptual según las reglas del DBMS escogido sobre cada máquina física traduciendo el esquema conceptual sobre cada DBMS.

Mi propuesta es simple:

El nivel lógico de la arquitectura distribuida se corresponde al Esquema Conceptual de la Base de Datos y el nivel físico al Esquema Interno. Las visiones de aplicación son ayudas y/o limitaciones de acceso por funcionalidad o derechos.

3. La clasificación clásica de los modelos de datos.

Con el inicio de los sistemas cliente/servidor aparecieron de forma natural tres modelos de datos que permitieron organizar las arquitecturas de los primeros sistemas.

3. 1. Datos centralizados.

Los datos se organizan en una única base de datos a la que accede todo el sistema distribuido. Fue el modelo inicial ya que al aparecer la necesidad de diseño Cliente/Servidor los datos en los HOST estaban organizados de esta forma.

Los datos centralizados tienen tres importantísimas ventajas:

- Son facilites de administrar.
- Es muy fácil mantener la coherencia de datos.
- Están permanentemente actualizados.

Pero en un sistema distribuido los clientes remotos son habituales. Curiosamente al principio de los tiempos eran siempre remotos. Posteriormente y con la aparición de los PC's y las redes aparecieron clientes locales. En cualquier caso, con los datos centralizados el tráfico de líneas es mucho mayor y el tiempo de respuesta muy penalizado frente al modelo HOST en que datos y procesados comparten máquina. Obviamente, el problema se agrava en los clientes remotos si además necesitan autonomía.

La pugna datos centralizados, fáciles de administrar pero con gran tráfico de líneas y con riesgo de perdida de disponibilidad, versus datos particionados y replicados, más difíciles de administrar pero con mucho menor tráfico de líneas, se convierte en el primer criterio y condicionante del diseño de la arquitectura del sistema distribuido.

Ese inconveniente adicional de los datos centralizados, en organizaciones que no tiene servicio continuo de HOST, (argumento clásico, pero… ¿queda alguno?) y que los clientes no pueden trabajar en periodos en que el HOST no funciona, puede condicionar totalmente el modelo u obligar a cambiar la oferta horaria de servicio del HOST.

Hay que hacer un comentario más. De hecho hay dos versiones del modelo centralizado:

- Los datos corporativos, localizados normalmente en el HOST y necesarios para toda la organización.
- Los datos departamentales, centralizados igual en un servidor del ámbito del departamento y sólo de interés dentro de él. La responsabilidad de los datos es del propio departamento. Observe que aquí es un modelo centralizado distribuido. Un verdadero trabalenguas.

De hecho, una solución muy habitual para aprovechar las ventajas de administración de los datos centralizados es una arquitectura de datos centralizada pero en la que se combina una gran base de datos corporativa, accedida local y remotamente, coexistiendo con modelos centralizados departamentales, que pueden incluir o no datos replicados.

3. 2. Modelo particionado.

En el modelo particionado los datos están distribuidos por varias bases de datos repartidas por la plataforma distribuida.

La realidad de los datos particionados, incrementada por los procesos de Upsizing, derivó en un cúmulo de posibilidades que llevó al límite del caos y la impotencia debido a que las plataformas de hardware y los productos de software no podían proporcionar las prestaciones de integración necesarias.

Las aplicaciones que necesitaron un modelo particionado tuvieron que creárselo incluyendo las herramientas de administración. La necesidad de garantizar artesanalmente la coherencia y confidencialidad de los datos hicieron del diseño de estas aplicaciones un trabajo divertidísimo pero complejo y caro.

Recordemos una vez más que los datos particionados son mucho más difíciles de programar y administrar pero que suponen mucho menos tráfico de líneas.

Partición *versus* centralización es finalmente, si no hay un condicionamiento operacional, un problema de costes: sume coste de software para crear y administrar el modelo particionado reste el coste de líneas y la mejora del tiempo de respuesta (bastante difícil de traspasar a dinero), pondere la importancia de que los datos estén alojados muy cerca del punto de mantenimiento (situación habitual en datos particionados) y escoja. Siempre que no tenga precondiciones, claro está.

3. 3. Datos replicados.

Para participar de las dos ventajas, apareció pronto en la historia C/S una solución: copiar los datos "estables" de las bases de datos centralizadas y montarlas sobre un modelo centralizado "cerca" de los clientes. Son los modelos de datos replicados.

Periódicamente habrá que actualizar la copia. Y eso, si el volumen de datos no es pequeño, cosa muy habitual por no decir segura, no podrá hacerse en la mayoría de los casos por copia. Habrá que utilizar un proceso incremental. Y entonces aparecerá el problema de garantizar la coherencia entre el origen y la replica. En fin, que todo son problemas.

La replicación sigue siendo un modelo muy utilizado pero el aumento de potencia de las plataformas, la mejora de las líneas y la aparición de Internet ha permitido retrotraer algunos modelos replicados de vuelta al "redil" centralizado.

Existen tres formas de replicar:

3.3.1. Copiar.

Los datos se copian tal cual. Hay una versión especial que es la copia selectiva en la cual solo se copian parte de los registros. Una opción habitual es que la copia sea por algún criterio de partición horizontal. Por ejemplo, en los PC's de un vendedor solo se copian sus clientes.

3.3.2. Sumarizar.

Acumular datos a un nivel superior. Por ejemplo, en lugar de copiar todas las facturas para seguir los resultados de venta creamos una entidad nueva en que a nivel de cliente estén el total de ventas por día y por producto.

3.3.3. Reorganizar.

Reconvertir el modelo de datos original para eliminar atributos no necesarios y/o juntar entidades. En una replica de la entidad artículos para ventas eliminamos todos los atributos producción y compras.

Una forma habitual de implementar una replicación es un entorno Data Warehouse del que le hablaré en seguida.

En cualquier caso, la replicación obliga durante la administración del sistema a:

- Definir políticas de replicación: cuándo, cómo y dónde.
- Auditorias periódicas para garantizar la coherencia y totalidad de los datos .Si es posible hacerlo, han de ser automáticas.

La importancia del modelo replicado y los problemas de diseño que conllevan serán ampliamente tratados dentro de los capítulos de diseño.

La clasificación que hemos conocido hasta aquí, permite hacer una primera modelización muy útil. Pero la realidad demostró rápidamente ser mucho más compleja y desbordó esta primera clasificación clásica de los modelos de datos.

Más adelante se hace una clasificación mucho más general y actual.

4. Integración de datos.

Problema complejo el resultante de integrar datos de aplicaciones creadas independientemente en un proceso de Upsizing. Veamos el por qué.

4. 1. Inconsistencias.

Cuando realice un proceso de este estilo y compare la misma entidad en las dos bases de datos se encontrará con problemas de dos tipos:

4.1.1. Sintácticos.

Campos que en una de las bases de datos son de un tipo en la otra son de otro. O siendo del mismo tipo, tienen formato diferente. Por ejemplo, después de un proceso de Upsizing de la aplicación de fábrica y aplicación de administración hay que hacer convivir dos ficheros de productos. El código de producto en fábrica es un string de 10 caracteres pero en administración es de 8. O la unidad de medida es la BD de la fábrica un real y en la de administración un entero.

4.1.2. Semánticos.

Dos campos con el mismo o parecido formato tienen contenidos semánticos diferentes. Así, la familia del producto en administración clasifica al producto según el mercado de venta y en fábrica según el proceso de fabricación.

4. 2. Modelos de datos resultantes.

Como resultado de un proceso de integración de datos se encontrará con dos modelos de datos:

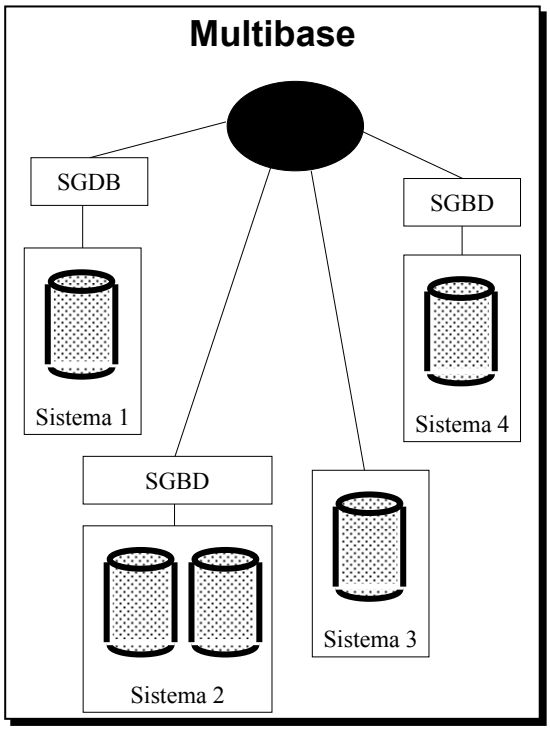


Figura 75. Multibase

4.2.1. Multibase.

La base de datos integrada conserva cada uno de los esquemas lógicos individuales y son los programas los que resuelven las diferencias semánticas y sintácticas.

La lógica de las entidades compartidas se encapsula en servidores que hace que a efectos de los programas clientes o servidores que los utilizan se comporten como una entidad única.

No es necesario que le remarque lo complicados y críticos que son los servidores de datos que encapsulan servicios de integración de datos, especialmente si las bases de datos están ligadas por comunicaciones. Piense en un proceso de modificación y note que habrá de actualizar todas las bases de datos y garantizar la coherencia.

4.2.2. BD Distribuida.

La base de datos integrada tiene un único esquema conceptual que permite ver todas las BD individuales como un solo nivel lógico. Cada base de datos integrada se corresponde un nivel físico. El diseñador habrá de conocer cuales de ellos son remotos para vigilar el rendimiento y los tiempos de respuesta.

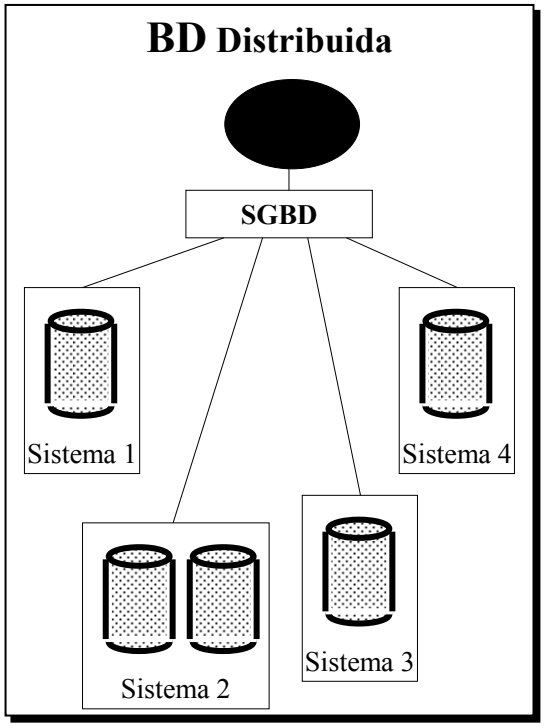


Figura 76. Base de datos distribuida

Es evidente que es mucho mejor trabajar con una BD distribuida que con una multibase. Sin embargo, en la práctica, los productos de software de BD que permitirían montar un esquema conceptual único son más del mundo de la investigación que de la industria.

Y si se resuelve por servicios, la diferencia entre multibase y BD

distribuida es tan pequeña que ambas soluciones prácticamente se confunden.

Observe, sin embargo que la terminología BD distribuida es muy peligrosa. Aquí se refiere a una base de datos con más de un esquema físico. El término no debe aplicarse a un modelo de datos distribuido, donde puede haber hasta ficheros secuenciales.

4.2.3. BD Federadas.

Es un caso especial de BD Distribuidas en que se pacta entre diferentes propietarios, independientes entre si, un esquema conceptual común plasmado en un acuerdo y resuelto muchas veces por vistas.

Cada base de datos ha de vincularse a la federación aceptando el acuerdo común.

4.2.4. Data Warehouse.

Si la BD integrada solo se necesita de consulta, una solución válida consiste en integrarla en un producto de Data Warehouse. A continuación presentaremos este tipo de producto de BD.

5. Terminología de Base de Datos.

La terminología de datos sólo puede definirse como caótica.

La necesidad de los departamentos de marketing de los fabricantes para vestir como nuevas soluciones productos ya existentes, mejorados eso si, ha llevado a una proliferación terminológica espeluznante.

He visto muchas clasificaciones y glosarios. Voy a intentar una sumarización de todos ellos. Insisto, solo intentar.

Me extenderé con mayor detalle en el Data Warehouse por su interés específico.

5. 1. Almacén de Datos (Data Warehouse).

Este termino lo oí por primera vez en el año 78. Desapareció. Reapareció de nuevo a principios de los 90 presentándose como la solución mágica a todos los problemas de datos del mundo C/S clásico. La euforia ha pasado y el término se ha ajustado a su justa dimensión.

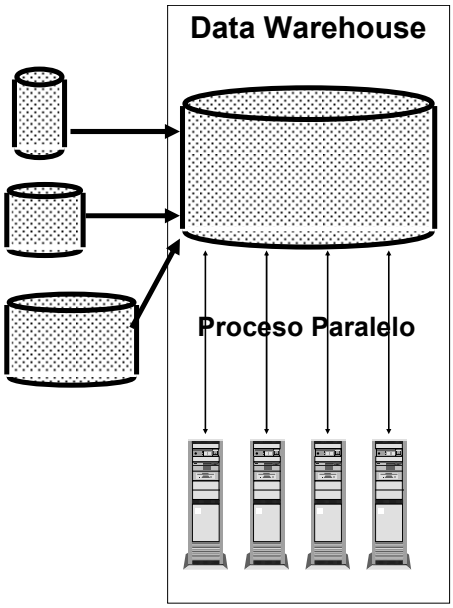


Figura 77. Data Warehouse.

Un Data Warehouse es una BD diseñada para soportar **consultas** para la toma de decisiones en una organización. Se actualiza por lotes y está estructurada para realizar consultas rápidas en línea y generar informes para los elementos de la organización, directivos o no.

Su función básica es la replicación y sumarización de datos corporativos estructurándolos y dejándolos disponibles para procesos de consulta.

Suelen contener grandes volúmenes de datos.

La deferencia entre un almacén de datos y cualquier otra BD operacional es que la información que contiene ha sido preparada para que las herramientas de acceso puedan trabajar eficazmente con ella. O al menos esa es la teoría. En la práctica, ¿me deja que le diga lo que realmente pienso? Un Data Warehouse es una gran base de datos replicada y sumarizada, con diseño convencional, más una herramienta de consulta sobre una plataforma muy potente que facilita el multicliente.

Se incluye soporte multimedia y herramientas para mantener un diccionario de datos a disposición del usuario. Y un mecanismo para obtener datos replicados directamente o de formatos heterogéneos permitiendo entre el proceso de extracción y carga ajustes de formatos. Este mecanismo le permite realizar una integración de datos cuando las fuentes son heterogéneas y resolver algunos de los problemas del Upsizing.

La política de replicación se establecerá periódicamente o en caliente. No utilice, si puede esta última opción.

Para la replicación pueden utilizarse:

- Herramientas ETL (extracción, transformación y carga) compradas a terceros.
- Procedimientos SQL.
- Programas específicos.

Otro aspecto de gran interés en un Data Warehouse es que la necesidad de rendimientos altos incorpora proceso paralelo tipo clúster en el motor lo que permite grandes volúmenes de transacciones y un número alto de clientes.

Es una buena solución cuando se realiza un Upsizing de consulta y aparecen de golpe un gran número de clientes nuevos. Aumentando la potencia del proceso paralelo se consigue una alta adaptabilidad a un coste razonable.

De hecho, un Data Warehouse supone una arquitectura de datos a tres niveles:

- En el **primero nivel** están los datos operacionales suportados por las bases de datos y los directorios con información no estructurada.
- En el **segundo nivel** se encuentran los elementos de integración: herramientas ETL, procedimientos SQL, programas específicos, el metaesquema y el almacén del Data Warehouse.
- En el **tercer nivel** están las herramientas de análisis de los datos y los programas del sistema distribuido que se aprovechan de la visión integrada que proporciona el metaesquema.

5. 2. Mercado de Datos (Datamart).

A las bases de datos organizadas para un departamento o función se las suele llamar Datamart en lugar de Warehouse. Sin comentarios. No usaré este término.

5. 3. OLAP o proceso analítico en línea.

También conocido como base de datos multidimensional, permite a los usuarios analizar rápidamente información que ha sido organizada en vistas multidimensionales y jerárquicas.

Por ejemplo, las herramientas OLAP se utilizan para llevar a cabo análisis de tendencias sobre ventas e información financiera, permitiendo a los usuarios profundizar dentro de grandes volúmenes de estadísticas de venta para aislar los productos más volátiles.

5. 4. OLAP multidimensional (MOLAP).

Productos OLAP tradicionales que resumen transacciones dentro de vistas multidimensionales antes de que estas sean utilizadas con lo que las consultas de los usuarios son rapidísimas.

Francamente creo que es un nombre de marketing para OLAP.

5. 5. OLAP relacional (ROLAP).

En realidad es un término aplicable a herramientas que permiten extraer datos y crear vistas multidimensionales utilizando sentencias SQL complejas que trabajan sobre BD relacionales tradicionales. Permiten crear consultas seudo-OLAP sobre la marcha.

5. 6. Minería de Datos (Datamining).

Tecnología sofisticada de búsqueda de datos que utiliza algoritmos estadísticos para descubrir patrones y correlaciones en los datos. De gran interés, pero fuera del ámbito de diseño distribuido.

Mientras que las herramientas OLAP permiten comparar, por ejemplo las ventas de dos trimestres, la minería permite realizar una búsqueda a través de todos los datos de ventas y luego presentar hipótesis a analizar.

Actualmente el termino minería se está usando muy alegremente en productos y situaciones que van sólo un paso más allá de la consulta comparada. Sin embargo, la verdadera minería de datos incorpora redes neuronales, árboles de decisión, inducción de reglas, y otras técnicas que como se puede observar, no son ninguna trivialidad.

5. 7. Operational Data Store.

Según W.H.Inmon es una recopilación de datos, actuales o casi actuales, orientados a temas, integrados y volátiles, que dan soporte a la toma de decisiones operativas (tácticas) del día a día de las compañías.

Requieren actualizaciones *on line* y *batch* diarias de la información.

5. 8. Sistema de soporte de decisiones (DSS).

Sistema de información y planificación que permite a los usuarios interrogar a las BD de una forma natural y cómoda (¡¿?!), analizar información y predecir el impacto de una decisión antes de tomarla.

Un DSS es un conjunto coherente e integrado de programas que comparten datos normalmente sobre un Data Warehouse,

A menudo pueden recuperar datos almacenados en fuentes externas (tema éste de gran interés) que pueden compararse y utilizarse para finalidades estadísticas e históricas.

5. 9. Sistemas de información para ejecutivos (EIS).

De todos los términos de análisis y soporte a las decisiones es el término más antiguo aplicado a las técnicas que consolidan y resumen datos en una organización. Normalmente el soporte es también un Data Warehouse.

Proporciona información a la dirección tanto de fuentes internas como externas.

Los EIS proporcionan algunas funciones de manipulación del tipo “qué pasaría si” de un DSS pero no verdaderas capacidades de crear modelos.

6. Acceso a los datos.

En el 99,999% de los casos el acceso a los datos es vía SQL. El SQL puede utilizarse directamente sobre las herramientas propietarias de las BD y a través de ODBC o ADO.

La gran ventaja de utilizar SQL estándar sobre ODBC y en menor medida sobre ADO es la transportabilidad. Los partidarios de utilizar la vía directa que proporcionan todos los motores le dirán que es mejor porque los estándares tienen perdida de rendimiento. Este argumento, que podía ser cierto en las primeras implementaciones del estándar ODBC, ya no es válido actualmente. El único

inconveniente que puede tener es operativo: es posible que su programa, el driver del compilador en que ha escrito su programa, el driver del sistema y su motor de BD tengan algún que otro problema de comunicación. Pero hoy día ya todo funciona de forma integrada rápidamente y a la primera.

Hay también herramientas de transacciones distribuidas para las que necesitará una formación especial, tanto en la teoría como en la práctica. Son potentes, pero caras y poco o nada estándares.

Finalmente, recuerde que puede utilizar vistas de la BD y procedimientos catalogados para sus propósitos.

7. El problema de la calidad de los datos.

Poder garantizar la calidad de los datos es fundamental para la supervivencia de la empresa: toma de decisiones erróneas, mailings a direcciones de correo que ya no son válidas, etc..

El problema de la calidad de los datos tiene, en un sistema distribuido, dos dimensiones:

- El volumen. Muchos datos potencian poca calidad.
- La distribución potencia la heterogeneidad.

Obviamente, la primera dimensión es general a cualquier sistema, la segunda es específica de los entornos distribuidos.

Las políticas y estrategias de diseño distribuido deben contemplar la calidad de los datos como objetivo prioritario.

En este apartado, la integración de datos será fundamental.

8. Clasificación actual de los modelos de datos en un entorno distribuido.

Como ya se ha dicho anteriormente, la realidad de los datos en una plataforma distribuida es mucho más compleja que simplificar los modelos de datos en centralizados o no.

Puede obtenerse una visión mucho más general enfocando el problema desde la perspectiva del programa a través del nivel lógico que proporciona el esquema conceptual y analizando, resolviendo y protegiendo los puntos de heterogeneidad proporcionados por el nivel físico marcado por el esquema interno y la localización de cada base de datos en la plataforma.

Observará en todo lo que sigue que no propongo ningún grupo que no sea sobre una BD relacional.

Ya he comentado que el modelo relacional es la única implementación de datos operativa en un entorno distribuido industrial.

Por otra banda, si en un sistema distribuido todavía tiene un punto de heterogeneidad resultante de una implementación de datos no relacional lo primero que tiene que hacer es valorar si puede cambiarla. Si es así, el problema será la

reconversión del software de acceso a los datos si la lógica de datos no está encapsulada. Vea que opciones tiene. Hay productos, por ejemplo, que le permiten utilizar aplicaciones COBOL creadas para ficheros IS sobre bases de datos sin tocar los programas. Si tiene una solución de este estilo, adóptela, utilice a partir de entonces SQL y poco a poco vaya realizando la migración.

Le propongo que trabajemos con el siguiente Modelos de Datos Extendido para cubrir todas las situaciones de una plataforma distribuida.

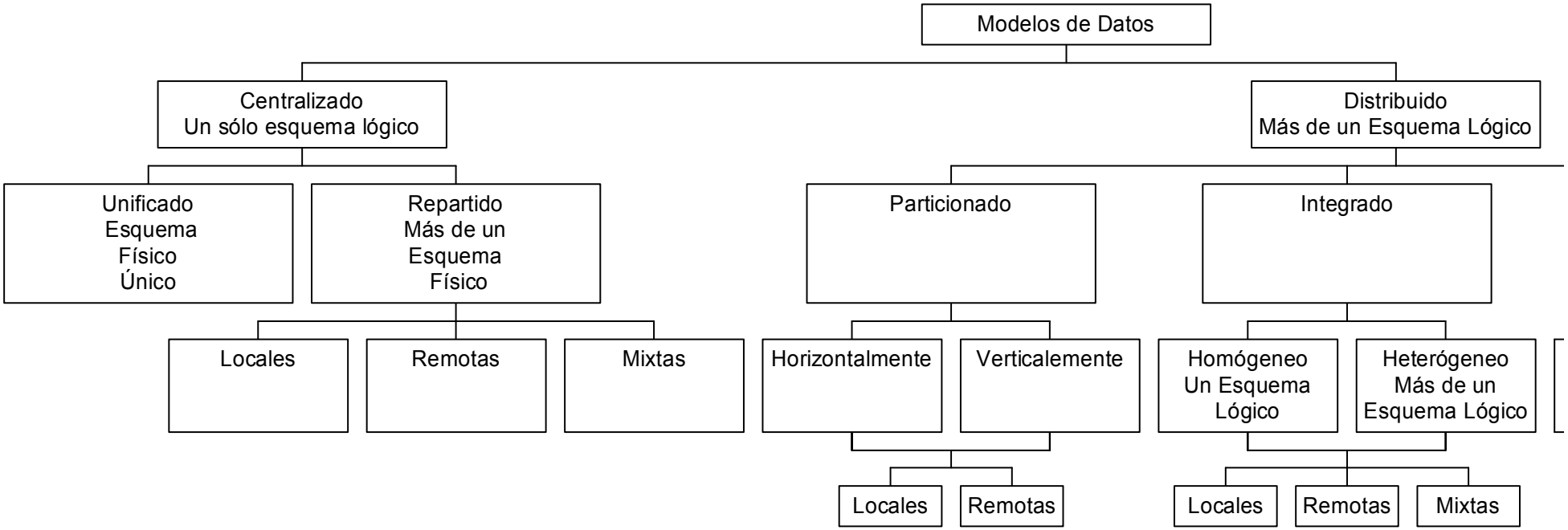


Figura 78. Modelo de Datos Extendido en un entorno distribuido

El primer nivel de clasificación se establece por el número de esquemas lógicos existentes en el modelo.

Se crean los siguientes grupos:

8. 1. Modelo Centralizado.

Hay un sólo esquema conceptual. Tienen dos subgrupos:

8.1.1. Unificado.

Hay un solo esquema interno. Se corresponde con el modelo centralizado de la clasificación clásica.

8.1.2. Repartido.

El esquema conceptual está repartido en más de un esquema físico. Hay que vigilar los problemas de coherencia y sincronización de las copias de seguridad. Si todos los esquemas internos son locales, el problema acaba aquí. Si alguno de ellos es remoto, los problemas de coherencia se agravan y hay un punto de heterogeneidad por comunicaciones lentas.

Si de la coherencia se encarga en gestor, a efectos de diseño equivalen a caso anterior unificado. Si no es así, estamos, siempre a efectos de diseño, en un modelo integrado homogéneo.

Observe que las base de datos federadas y las bases de datos distribuidas (en el concepto anterior) van en este bloque.

8. 2. Modelo Distribuido.

Hay más de un esquema conceptual. Tienen dos subgrupos:

8.2.1. Datos Particionados.

Las diferentes entidades de datos están repartidas por la plataforma. Evidentemente aquí también existe la combinación de esquemas internos locales o remotos.

Existen dos posibilidades de particionar los datos:

8.2.1.A. Verticalmente.

Unas entidades están repartidas por los diferentes esquemas conceptuales sin duplicidades. Por ejemplo, todos los productos en un sitio y todos los clientes en otro.

8.2.1.B. Horizontalmente.

Una o varias entidades están fraccionadas en más de un esquema conceptual. Por ejemplo, los clientes de Barcelona en Barcelona y los de Girona en Girona.

8.2.2. Integrados.

Vale aquí todo lo dicho en el apartado anterior dedicado a este tema.

Recuerde que los datos integrados pueden ser:

8.2.2.A. Homogéneos.

Un sólo esquema conceptual. Es la única excepción al criterio de que en los datos distribuidos hay un solo esquema conceptual. Se corresponde básicamente con el modelo centralizado repartido. Se conservan los dos nombres para saber cual es el origen de los datos. Aquí, upsizing, allí metamodelo.

8.2.2.B. Heterogéneos.

Más de un esquema lógico.

Y que existe también la combinación local y remota.

8. 3. Modelo Replicado.

Vale también aquí recordar lo que hemos hablado anteriormente sobre datos replicados.

Recuerde que hay un solo esquema conceptual, un solo esquema lógico y que el origen de los datos registrados son otras bases de datos.

Normalmente son datos locales al entorno donde se usan contenidos en un servidor de la red.

Y recuerde también que hay tres grupos:

- Copia.
- Sumarización.
- Reorganización.

No hace falta decir que la arquitectura de datos de la plataforma distribuida se construirá como combinación de uno o más de estos modelos de datos.

9. Procedimientos Catalogados.

9. 1. ¿Qué es un procedimiento catalogado?

Un procedimiento catalogado es una colección, etiquetada con un nombre, de comandos con lógica de datos que es compilado, verificado y almacenado en el servidor de datos.

El procedimiento se llama a través de un mecanismo síncrono parecido al de "Remote Procedure Call" (RPC).

La base de datos lo trata como cualquier otro objeto y lo registra en su catálogo. El acceso es gestionado a través de los mecanismos de seguridad del motor de la base de datos. Los procedimientos catalogados viajan con el esquema de la base de datos.

9. 2. Interés de los procedimientos catalogados en el diseño distribuido.

¿Por qué son tan interesantes los procedimientos catalogados en un entono de diseño distribuido?

Para entenderlo fijémonos en la figura. Imaginemos que ha de hacer un proceso de lógica de datos que supone varios accesos SQL.

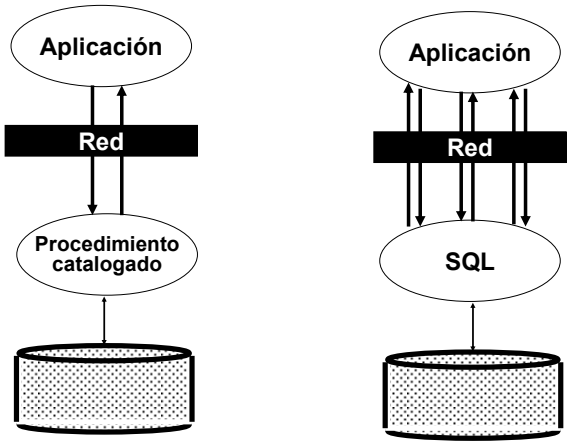


Figura 79. Procedimiento Catalogado versus SQL

Si utiliza el sistema habitual cada vez que realice un paso SQL la petición y los datos viajarán a través de la red. Si encapsula la lógica de datos en un procedimiento catalogado, por la red solo viajará una petición y una respuesta. Toda la lógica da datos se ejecutará localmente en el servidor.

Imagine, por ejemplo, que quiere dar de baja un producto de venta. Deberá comprobar que no hay stock, que no está utilizado en ningún pedido vivo, que no hay ninguna fórmula de fabricación ligada, etc. Compare el flujo SQL necesario para las comprobaciones y las bajas y la ejecución de un único procedimiento catalogado.

La gran ventaja de los procedimientos catalogados es, pues, el **rendimiento.**

También es importante el hecho de que, como hemos dicho, **viajen con la base de datos**. Aunque esto, generalmente una ventaja, puede ser en algunos casos un inconveniente.

El acceso a un procedimiento catalogado desde un cliente o un servidor es síncrono, como una rutina convencional, incluyendo en la llamada los parámetros de entrada necesarios. La llamada origina la ejecución del

procedimiento en el servidor de la BD consiguiendo así un mejor rendimiento (perfomance).

9. 3. Utilización.

La utilización genérica de los procedimientos catalogados es encapsular lógica de datos en el servidor de datos de la aplicación.

Se pueden encapsular entre otras cosas:

- Reglas de negocio críticas.
- Reglas de negocio con muchas SQL's.
- Reglas de negocio confidenciales.
- Select's de ámbito muy amplio.
- Reglas de integridad.
- Funciones de administración y mantenimiento de la BD.

Es decir, que los procedimientos catalogados pueden convertirse en servidores de servicios disminuyendo el peso de los clientes y mejorando significativamente el rendimiento.

Un valor añadido que no hay que despreciar es el hecho de que si el esquema conceptual cambia nada más hay que cambiar el procedimiento catalogado ya que forman parte de la propia base de datos.

Administración del Sistema Distribuido.

1. Introducción.

Como ya se ha comentado anteriormente, una de las cosas que diferencia claramente un sistema distribuido de uno centralizado es la administración del sistema. En general, los sistemas distribuidos son más difíciles de administrar que los que no lo son.

Recuerde que el componente del Middleware que incluye las funciones de administración es el DSM. (*System Distribuited Management*), que ya le he presentado anteriormente

En este capítulo haremos un primer balance y aproximación a la gestión de la administración de los sistemas distribuidos dejando para la segunda parte hablar de políticas y estrategias de diseño.

2. ¿Qué es diferente en la administración de un entorno distribuido?

Existe una comparación clásica que incluyo junto a otros puntos más actuales en la siguiente tabla. Le he subrayado los puntos que, a mi juicio, son de especial interés y complejidad. Y algunos comentarios totalmente subjetivos.

Proceso	Sistema no Distribuido	Sistema Distribuido	Observaciones
Usuarios	Entrenados	Poco entrenados	La clasificación clásica afirma que en el entorno distribuido los usuarios están peor entrenados. Ello viene a conllevar mayores problemas con los usuarios. No estoy de acuerdo. Las interfícies gráficas y los estándares de operación han estandarizado la operativa. Siempre, claro está, que el diseñador se aproveche de ello.
Datos	Fácil	Difícil	La consistencia y protección de datos es más difícil si la arquitectura de datos tiene componentes no centralizados. Punto claro a favor de los no distribuidos.
Back-up de datos	Fácil	Difícil	Evidentemente, siempre que el modelo de datos no sea centralizado.
Autentificación y Seguridad	Muy fácil	Difícil	Tanto más difícil cuanto más heterogéneo sea el sistema.
Back-up de procesos	Muy difícil	Difícil	Personalmente creo que este apartado sobra. Hablar de Back-up de procesos en un entorno HOST que no se cuelga en la práctica nunca es hablar por hablar.
Sistemas Operativos	Uno	Más de uno	Si hay homogeneidad, como pasa generalmente por la omnipresencia de

			Windows, como si tuviera uno sólo. Aquí el problema puede ser de otro tipo: la robustez de un sistema de HOST frente a Windows.
Tipos de Sistemas Operativos	Propietario	Abiertos	En los abiertos, casi todo Microsoft con presencia lentamente en alza de LINUX, especialmente en servicios de comunicaciones.
Nodos Disponibles	Uno	Más de uno	Este apartado sobra. La posibilidad de localizar los servicios en más de un nodo solo tiene sentido en sistemas distribuidos.
Rendimiento	Conocido	Difícil de Evaluar	Evaluar el rendimiento de un sistema distribuido se puede hacer por los recursos del monitoring (seguimiento de la explotación). Lo difícil es analizar donde están los problemas.
Problemas	Centralizados	Dispersos	Dispersos no quiere decir necesariamente más difíciles.
Tolerancia a fallos	Baja	Alta	No estoy de acuerdo. Si se cuelga un HOST se para todo. Pero un HOST no se cuelga en la práctica nunca. Los sistemas distribuidos necesitan pasar los servicios caídos a otro nodo
Proveedores	Muy Pocos	Muchos	Puntito a favor de los distribuidos.
Integración componentes	Trivial	Fácil	En el fondo, si el diseñador tiene calidad....
Costos	Altos	Bajos	Decir bajos me parece excesivo. Mejor sería decir menores.
Facilidad de Crecimiento	A saltos	Modular	Punto a favor de los distribuidos
Coste del Crecimiento	Alto	Bajo	Punto a favor de los distribuidos
Diseño del DSM	Innecesario	No trivial en sistemas avanzados	Punto a favor de los centralizados.
Fiabilidad de la plataforma	Total	Conflictiva en relación inversa a la inversión realizada	Punto claro a favor de los centralizados
Adaptabilidad	Baja	Altísima	Gran punto a favor de los distribuidos

3. ¿Qué se ha de gestionar en un sistema distribuido?

La gestión de un sistema distribuido se centra en cinco ámbitos:

- Hardware.
- Software de base, aplicaciones y servicios, aunque en la práctica el primero que es generalista, suele diferenciar de los otros dos que necesitan una cierta especialización.
- Conectividad.
- Datos.

❍ Personas, elemento evidentemente no técnico pero fundamental.

En el siguiente cuadro se muestra un esquema de los elementos, parte superior, y funciones, parte inferior, más importantes de los cuatro primeros.

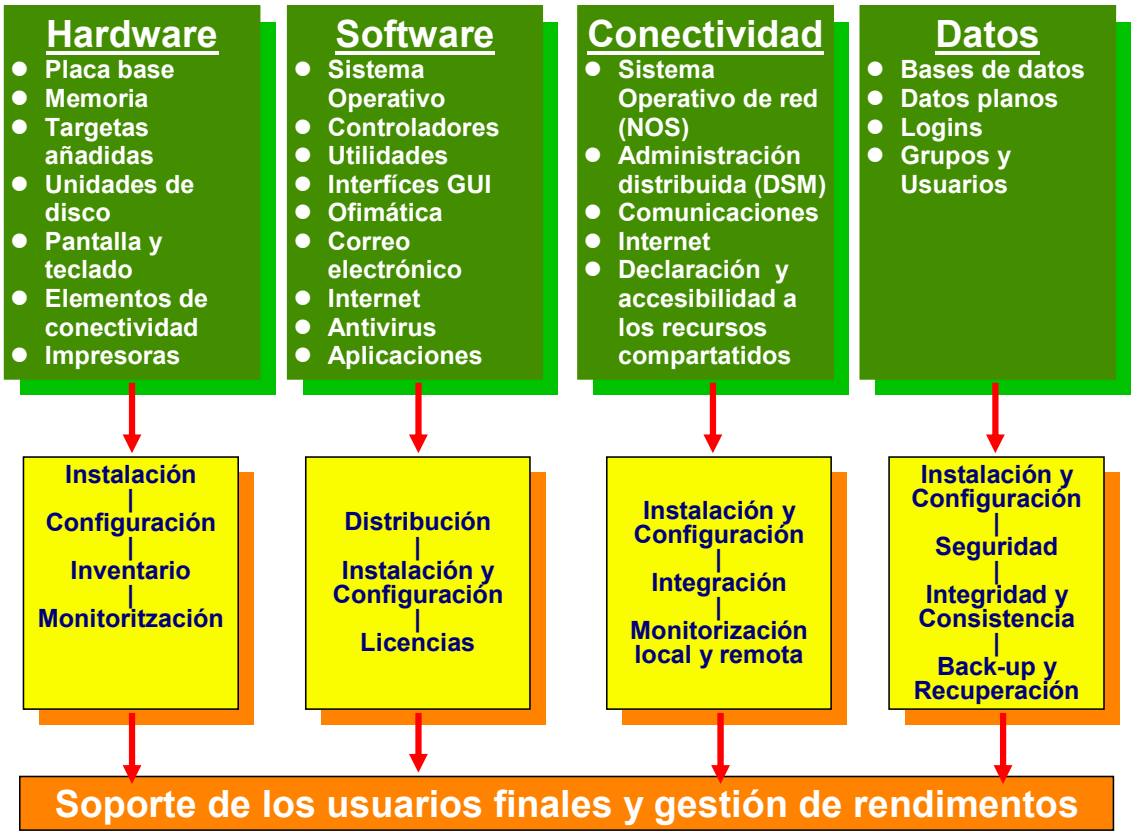


Figura 80. Componentes a Gestionar en un Sistema Distribuido

En la segunda parte del libro, dedicada al diseño ya haremos la relación más exhaustiva. Pretendo aquí introducir únicamente los conceptos generales y la problemática.

4. Procesos básicos de un DSM.

Incluyo a continuación una relación introductoria de los factores que influyen en cada uno de los procesos que a mi juicio son básicos en un DSM.

4. 1. Referenciar los recursos: Naming and Directory.

Permítame respetar la terminología inglesa que es muy clara. Cada recurso del sistema debe tener un nombre lógico y estar catalogado

4. 2. Situación y localización.

Cada recurso tendrá un estado (funcionando, estropeado, apagado, protegido, etc.) y una localización física en el sistema distribuido.

4. 3. Datos y procesos de protección de la información.

Hablamos de los procesos de seguridad.

Si hay más de un esquema conceptual, más de un motor y hay dispersión física puede haber problemas serios para la autentificación única de logins y usuarios.

4. 4. Integridad y Consistencia de los Datos.

Los factores anteriores hacen que muchos productos de datos de Middleware no sean capaces de garantizar la integridad (que estén todos los datos) y la consistencia (que los datos cuadren entre ellos). Los programas deberán asumir parte de estas funciones.

Hay necesidad de crear programas de **Auditoria de Datos** que permitan garantizar la integridad y la consistencia verificando restricciones de integridad, las replicaciones y cruzando las informaciones. Estos procesos son específicos de cada aplicación y deben incluirse en el diseño de administración del sistema.

4. 5. Back-up y recuperación.

Hacer las copias no es lo complicado, lo difícil puede ser sincronizarlas. En datos distribuidos, y en menor medida en replicados, recuperar una parte sin recuperar el total puede ser muy difícil. Y recopilarlo todo puede ser tan costoso en tiempo que haga la solución inviable sin una perdida no admisible de servicio. Y si la recuperación no es por re-copia, después de recuperar habrá que pasar los programas de auditoria.

4. 6. Inventario de elementos.

Si alguna vez tiene que responder a una pregunta del tipo: ¿qué costará subir la versión del paquete de ofimática? que Dios le ayude. Es la pregunta de todas las pesadillas.

Deberá conocer que usuarios la utilizan y de aquí calcular cuantas actualizaciones de licencia necesita. Esta pregunta aún es fácil. Pero seguro que en un cambio así deberá subir el nivel medio de la plataforma: CPU, memoria y disco. Que hardware está preparado y cual no, que hay que cambiar en cada PC no preparado y hasta si es posible hacerlo, cuantas horas necesitará dedicar a cada ordenador, le será, créame, prácticamente imposible de conocer. Si es afortunado, solo podrá dar una cifra orientativa.

De cualquier forma, para hacer el intento deberá llevar un mínimo inventario de elementos hardware y software.

Son factores determinantes del inventario.

- Existencia de elementos muy diferentes, dispersos y poco controlables.
- Costes de elementos tan bajos que no justifican los costes de inventariarlos.
- Situación y estado de los elmentos en organizaciones de unidades operativas que compran y mantienen de forma autónoma.
- Personalizaciones del tipo plantillas, macros, etc.., nunca inventariadas y que pueden “desaparecer” o dejar de funcionar con el cambio de versión.
- Desconocimiento de la existencia de productos instalados por el propio usuario.

Ocurre en muchos casos que el coste de tener el inventario al día es mayor que el beneficio tangible ya que es muy difícil evaluar en dinero las ventajas de control y información que puede suponer llevar bien el inventario.

Hoy día pueden hacerse inventarios remotos y en caliente, tanto de hardware como de software, pero, ¿son fiables y completos?

4. 7. Instalación inicial y cambios.

Influyen en esta actividad:

- La dispersión geográfica.
- La dispersión de elementos.

A pesar de la existencia de estándares, “enchufar” los sistemas no es tan rápido y fácil como se vende. En ocasiones, incluso muchos productos se estorban entre sí.

4. 8. Configuración.

Puede ser estática o dinámica. Influye en ella:

- La dispersión de entornos.
- La dispersión geográfica.
- Las acciones realizadas por el propio usuario.
- Disponibilidad de los servicios y/o elementos.

Es un problema a repartir entre el DSM y la aplicación. Hay que identificar los puntos críticos, decidir los procesos de recuperación y escoger si la activación es automática o manual.

Un factor importante es la dispersión de horarios y días laborables si hay fuerte dispersión geográfica.

4. 9. Rendimiento.

Difícil de prever y complicado de analizar por la acumulación de factores:

- Potencia de la plataforma de hardware.
- Velocidad y congestión de la conectividad.
- Rendimiento de los productos y capas de Middleware.
- Calidad de las aplicaciones.

Contestar a la pregunta: ¿Donde esté el problema? puede ser imposible.

4. 10. Soporte a usuarios.

Hablamos de cuando, donde, de que forma y con que coste pueden los usuarios obtener ayuda delante de un problema de explotación.

El soporte a usuarios incluye:

4.10.1. Formación y ayuda.

En el momento inicial y en cualquier otro momento en que el usuario necesite conocer las posibilidades de las aplicaciones y la forma de obtenerlas.

4.10.2. Gestión de incidencias y problemas.

- Organización de la Hot-Line : ¿es necesaria? ¿es rentable? ¿quien la da?
- Diagnostico: ¿qué pasa? ¿cual es la causa?
- Resolver el problema: ¿quien ha de actuar? ¿ya está resuelto?

4. 11. Mantenimiento del software.

Factores:

- Política y gestión de los servidores de programas.
- Gestión de Licencias.
- Gestión de Versiones.
- Gestión de objetos distribuidos.

4. 12. Antivirus y Firewall.

Hay que instalar un vigilante y tener un detector y eliminador siempre al día. La vigilancia ha de estar siempre arrancado y abarcar todos los puntos de entrada incluidas las disqueteras y Internet.

Cuando tenga una alarma de virus en un punto, pare y revise todo el entorno potencialmente contaminado.

4. 13. Autentificación de usuarios.

Vale lo dicho para los datos.

4. 14. Protección de recursos.

Previa autentificación del usuario, por su código.

Es importante instalar Firewall en las entorno en Internet.

4. 15. Puntos especialmente preocupantes.

En particular, preocúpese especialmente de:

- Establecer una política de soporte a usuarios y, en particular de resolución de problemas.
- Intentar avanzar en la gestión de rendimientos.
- Buscar buenas soluciones en entornos muy heterogéneos.

5. Situación de los productos de DSM.

La administración del sistema distribuido tiene la opción de utilizar productos DSM que amplían y complementan las posibilidades DSM del Middleware que se proporciona con los NOS.

Básicamente hay dos bloques de productos:

- Extensiones de los productos DSM que se incluyen en el NOS. Suelen estar desarrollados por los mismos fabricantes del NOS. Su utilidad práctica se limita a aquellos entornos muy "homogéneos en ese fabricante".
- Productos de terceros pensados y construidos con filosofía integradora. Suelen ser de calidad ya que han de incluir muchos fabricantes y han de aportar soluciones reales.

Se hace difícil hacer una relación de productos. Además no es mi objetivo. Sin embargo, dada la importancia que estoy convencido tienen para conseguir costes razonables en una buena aplicación distribuida con problemas complejos de administración, voy a presentarle alguno de los criterios que ha mi juicio deben mirarse en un proceso de selección de este tipo.

El gran inconveniente es el precio. Tendrá que tener un entorno muy amplio para que se le justifique la inversión en un producto de este tipo.

¿Qué se ha de buscar cuando se está haciendo una selección de un producto de DSM? Valore:

- **Adaptación a los sistemas propios**. El argumento principal.
- La posibilidad real de **un punto centralizado de gestión**. Como veremos en diseño, fundamental
- Reparación automática de configuraciones de hardware y software.
- Un repositorio de datos con los elementos del sistema distribuido y, si es posible, accesible por API's.
- Mecanismos de gestión de informes.
- Alarmas y avisos.
- Herramientas locales y, sobre todo, remotas de diagnostico y análisis.
- Herramientas para la recuperación remota de dispositivos.
- Facilidades para el control y distribución de versiones de software.
- Tecnologías de detección y eliminación de virus.
- Recursos de inventario automático.
- Universalidad. ¿le sorprender ve tan abajo este punto? No debería ser así. Tenga en cuenta que cualquier producto de este tipo incluirá de base todas las plataformas habituales

6. Introducción al Diseño de la Administración de Sistema.

Recuerde que las funciones a cubrir en un sistema distribuido son básicamente las mismas que en uno que no lo es. El diseñador deberá conocer cual es la problemática general de la administración del sistema. De ella, cual afecta a su aplicación. De aquí, relacionar las necesidades de administración y marcar las propuestas, que deberá concensuar con el Administrador del Sistema y, si conviene, con los usuarios. Surgirá así el Plan de Administración donde se resumirá la **política de administración** decidida para el sistema distribuido..

A partir de aquí, deberá analizar cuales de estas necesidades quedan resueltas por el DSM. Y finalmente, implementar como parte de su aplicación las que falten. Y siempre con la idea de que el que vaya detrás pueda reutilizar al máximo las nuevas funciones.

A continuación deberá identificar las tareas de trabajo intensivo y definir procedimientos automáticos para ayudarlas. Este factor será fundamental en minimizar costes.

Finalmente pensar en la centralización del control de la gestión y decidir como será el cuadro de mandos de explotación.

Todas estas nuevas funciones se agrupan, como ya hemos comentado anteriormente, en el Diseño de la Administración del Sistema.

Las funciones a cubrir son prácticamente iguales en un sistema distribuido homogéneo de uno heterogéneo. La diferencia está en la dificultad de implementar las soluciones. La dificultad aumenta muy progresivamente cuanto más heterogéneo sea el sistema.

Como ya hemos dicho, cuando el diseñador aborde esta parte, deberá tener muy en cuenta los criterios de reutilización ya que las funciones de DSM que implemente serán muy probablemente de interés para futuras aplicaciones. Recuerde que a la hora de fabricar componentes reutilizables no hay que perder de vista el ratio prestaciones/costes y caer en el extremo de generar un software tan general que en su instalación no se llegue a amortizar nunca.

Una opción alternativa a implementar las funciones de administración de las que no se disponga en el DSM del sistema es comprar un paquete de administración. El problema es que la solución es cara. Deberá comparar su coste con el de desarrollo. Como criterio general si puede cómprese un paquete.

Pero recuerde que comprar la solución no le libera de analizar previamente las necesidades y marcar las políticas de administración de su aplicación distribuida.

Estándares.

1. Introducción.

A lo largo del camino que llevamos recorrido ya se ha hablado directa o indirectamente de los estándares.

Es un buen momento para hacer una recapitulación y algunas reflexiones.

2. Los estándares.

La necesidad de estándares es evidente. Es la única forma de conseguir Middleware en su sentido idílico.

En el mundo del desarrollo distribuido se ha avanzado mucho. Y aunque organizaciones sin ánimo de lucro han hecho muchas propuestas, los estándares alcanzados han sido mayoritariamente “de facto”: ODBC, TCP/IP, OLE, Active X, J2EE, XML y un largo etc.

En el mundo del NOS + DSM, la situación es mucho menos clara pero parece decantarse por las líneas Windows o Linux para el NOS y por las propuestas de OSF o Microsoft para el DSM (recuerde el capítulo anterior).

De cualquier forma, este es un mundo en perpetua evolución en el cual se hace difícil hacer una relación escrita y actualizada de los estándares. No voy a caer en el error. Además no es mi objetivo en este libro dedicado al diseño y no a la infraestructura distribuida.

3. Estándares e implementación.

Un mismo estándar puede ser fabricado por más de un vendedor, sobre todo cuando está disponible en plataformas diferentes.

Cada vendedor hace una implementación del estándar. La parte básica del estándar suele ser respetada. Pero el fabricante, para diferenciarse en calidad de la competencia, suele añadir prestaciones extendidas. Usarlas suele ser tentador, pero muy peligroso. Si añade una nueva plataforma e incorpora un nuevo proveedor del estándar, no tendrá compatibilidad. Yo le recomiendo no hacerlo. Recuerde sobre este punto la reflexión que sobre MS SQL Server hemos hecho antes.

Los problemas generados por la presencia de más de una implementación del estándar generan problemas semánticos y sintácticos idénticos a los que aparecen por la existencia de más de un estándar en el mismo ámbito de servicios.

Y si no hay más remedio que convivir con ello, la solución pasa por encapsular.

Para aplicar técnicas de encapsulamiento en este ámbito es recomendable actuar así.

- Definir API's de compañía con contenido semántico y sintáctico perfectamente definido.
- Traducir este contenido a las API's de cada producto.
- Le recomiendo que las API's de compañía no incorporen contenido semántico no presente en los productos

De hecho, estas son técnicas de implementación de Middleware real (del cual hablaremos en los siguientes capítulos).

Hay dos formas de definir las API's de compañía:

- Coger como modelo semántico y sintáctico uno de los productos, preferentemente el más "estándar".
- Definir una semántica y sintaxis propias.

Es recomendable, sin lugar a dudas, la primera vía. Y obligada si uno de los productos es ya un estándar "de facto".

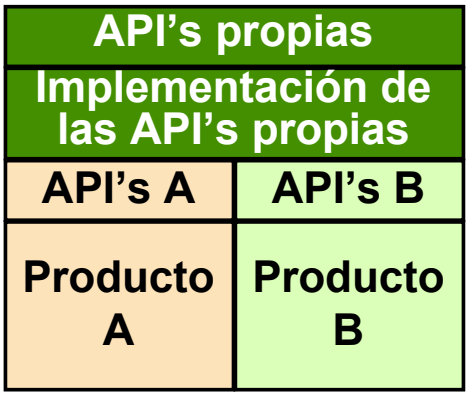


Figura 81. Encapsulamiento de Middleware

4. Recomendaciones finales.

Esté muy atento a la evolución de los estándares. Para conseguirlo, lea revistas, asista a congresos, vaya a presentaciones de productos de diversos fabricantes. **Y no se olvide de intercambiar experiencias con otros profesionales.**

Si está muy ligado a un fabricante, escuche sus recomendaciones. Y si tiene la posibilidad de escoger decántese, naturalmente, por la opción más estándar.

Introducción a la Comunicación Cliente/Servidor.

1. Introducción.

Como parte del modelo distribuido habrá que decidir como se comunican los servidores con los clientes u otros servidores.

El tema es suficientemente importante como para que se trate a fondo dentro de la segunda parte dedicada específicamente al diseño.

Sin embargo, en esta primera parte introductoria es fundamental sentar las bases y que Vd., amigo lector, comience a familiarizarse con un tema tan básico.

Para facilitar la lectura, voy a referirme como cliente al elemento que pide el servicio, independientemente de si en esa posición hay un verdadero cliente o un servidor que temporalmente se convierte en cliente de otro.

Los modelos de comunicación, no lo olvide, tienen dos partes:

- La sintaxis y los formatos de comunicación que dependen de los productos.
- El transporte en si, que proporciona el transportista, elemento tan estandarizado e integrado en el Middleware que hoy día es, creo que afortunadamente, desconocido por la mayoría de desarrolladores.

2. Tipología de la interacción entre un cliente y un servidor.

Es bueno hacer una pequeña reflexión y recordatorio de lo presentado hasta ahora sobre la interacción entre los programas, clientes y servidores, de un entorno distribuido.

No hay que olvidar en ningún momento que esta interacción presenta las siguientes características básicas:

2. 1. Cooperativa.

Clientes y servidores se reparten el trabajo.

2. 2. Transaccional.

El cliente prepara una petición de servicio, la pasa al servidor quien la ejecuta y devuelve la respuesta. Recuerde que el modelo C/S obliga a que el mecanismo solo tenga una petición / respuesta por servicio solicitado. Diremos que la comunicación entre el cliente y el servidor es transaccional en este sentido, de autonomía e independencia de cada intercambio..

2. 3. Sincronizado o no.

Cliente y servidor tienen que sincronizarse o no en función de las necesidades de la aplicación y de las posibilidades del modelo de comunicación.

Tratemos a continuación este aspecto.

3. Comunicación síncrona, asíncrona y desacoplada.

3. 1. Comunicación síncrona.

En comunicación síncrona, el cliente lanza la petición y se para a esperar la respuesta.

Es el modelo convencional de comunicación entre un programa y una rutina. Por eso los modelos de comunicación síncronos se ha utilizado tanto.

En principio, puede parecer que no se necesita más para trabajar. **Gravísimo error** ya que hay una diferencia básica de la que ya hemos hablado en otro momento. En un programa clásico, cuando se llama a una rutina, si no llega respuesta con el programa en explotación, el problema que tendremos entre manos será mucho más grave que no el hecho en si; probablemente la máquina esté “colgada” o tenga graves problemas.

En un modelo de comunicación síncrono C/S la respuesta puede no llegar por problemas de acceso o de máquina en la localización del servicio, pero el programa cliente continua perfectamente activo esperando la respuesta.

No hay que decir que un modelo de síncrono necesita un mecanismo de multicliente para que varios de ellos puedan realizar llamadas al mismo tiempo, aunque se atiendan una detrás de otra.

Hay dos submodelos de comunicación síncrona:

3.1.1. Síncrono no controlado.

La caída del servicio provoca el “cuelgue” del cliente. Afortunadamente no queda prácticamente ningún modelo síncrono de este grupo.

3.1.2. Síncrono controlado.

El Middleware realiza dos controles de existencia del servicio:

3.1.2.A. Control de conexión.

Cuando el cliente pide el servicio se comprueba que este disponible y en caso contrario se devuelve error inmediatamente al cliente.

3.1.2.B. Control de respuesta.

Normalmente por time-out, se controla que, si pasado un tiempo no hay respuesta, se devuelva control y aviso de error al cliente.

Hay dos modalidades:

- Esperando. Si el servidor no esta disponible, es decir "esperando", se devuelve el mensaje. Equivale a time_out=0.
- Cronometrado es el caso de time_out>0.

3. 2. Asíncrona.

En comunicación asíncrona, el cliente lanza la respuesta, continua trabajando y cuando le conviene recoge la respuesta.

Inmediatamente se observa que este modelo de comunicación es más potente que el síncrono, pero que también es más complejo de gestionar. Cada mecanismo que proporcione un modelo de comunicación asíncrono ha de proporcionar como mínimo tres mecanismos adicionales:

3.2.1. Multipetición.

Los clientes deben poder dejar peticiones para los servidores aunque estos estén ocupados. Este mecanismo, aunque escondido, también existe en la comunicación síncrona.

3.2.2. Interrogación.

Preguntar desde el cliente si la respuesta está disponible.

3.2.3. Almacén de respuestas.

El servidor ha de ser capaz de poder almacenar respuestas hasta que el cliente este disponible para recogerlas. De otra forma, el servidor daría la sensación de "cuelgue".

3. 3. Desacoplada.

En comunicación desacoplada, el cliente lanza la petición y no recoge nunca la respuesta ya que supone que el servicio se ejecutará siempre y acabará sin problemas.

Un ejemplo clásico es el uso de un servidor de impresión. El programa imprime por una impresora lógica y se despreocupa de si el spool está activado o no.

La filosofía de traspaso de datos entre aplicaciones (interfase) es también un modelo de comunicación desacoplado.

El modelo desacoplado es considerado por muchos autores como un modelo asíncrono límite. Yo particularmente pienso que no es C/S sino una comunicación por interfase ya que necesita un almacén intermedio de comunicación. Y el hecho de que eso se consiga en muchos casos con una cola, la aparición de un fichero en un directorio o de un registro en una entidad de la base de datos no cabía nada.

En otras palabras, obtiene un servicio sin utilizar comunicación cliente/servidor.

4. Esquemas de petición respuesta.

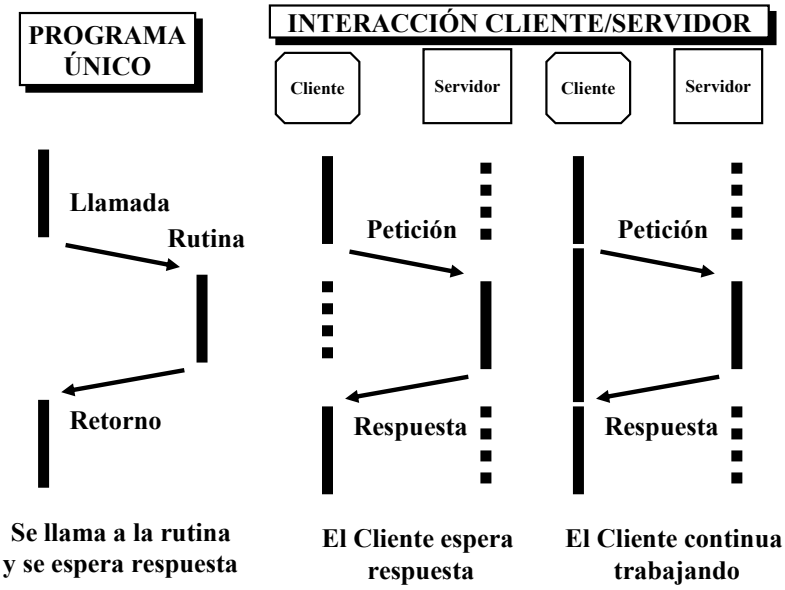


Figura 82. Comunicación C/S síncrona y asíncrona.

Observe la figura. A la izquierda tiene un esquema del comportamiento Call-Return de llamada a una rutina linkada en un programa.

En el centro tiene el esquema de una comunicación síncrona en la cual un cliente llama a un servidor y se espera a la respuesta. Las dos situaciones descritas parecen iguales. Pero si se observa bien hay dos diferencias notables: dos programas comunicándose y la posibilidad, ya comentada de que no llegue la respuesta. El servidor está arrancado y a la espera de la petición o atendiendo las de otros clientes. El cliente sigue activo y en ejecución pero esperándose a la respuesta. El tiempo de proceso del cliente mientras espera se pierde.

Observemos el esquema de la derecha de la figura en la que se muestra un modelo de comunicación asíncrono, en el cual el cliente continúa trabajando mientras se procesa su petición. La situación es notablemente diferente:

En una comunicación asíncrona el cliente queda liberado para hacer otras cosas mientras el servidor procesa su petición.

La utilización de este tiempo puede marcar la diferencia entre un buen diseño y otro mediocre. Déjeme ponerle un ejemplo.

Imagine que tiene una aplicación de captación de pedidos distribuida y que la está ejecutando en una delegación de ventas de Girona. Por necesidades de control de riesgo debe comprobar el crédito del comprador sobre una BD de Barcelona. Y la comunicación remota que tiene no es rápida, por amplitud de línea o por

congestión. Lógicamente la posibilidad de que el comprador sea moroso es muy baja (sino ya puede cerrar la delegación).

Puede diseñar su proceso así:

- Pedir los datos del comprador.
- Lanzar la petición remota la servidor de comprobación de crédito de Barcelona
- Continuar registrando los productos que el comprador necesita.
- A final de este proceso recoger la respuesta a su petición de comprobación de riesgo. Obviamente, si no ha llegado todavía se habrá de esperar.

Como normalmente la respuesta será que se podrá vender, habrá optimizado su proceso. Imagine la mejora del tiempo de servicio que conseguirá y las colas de clientes que puede evitar.

Pero si la comunicación no tuviera nada mas, el cliente debería saber en que momento el servidor está libre para comunicarse con él. Ya se ve que trabajando así el conjunto no sería operativo. Ha de existir un mecanismo que permita al cliente lanzar la petición sin saber el estado del servidor. Este mecanismo es el que en la figura se representa como la **lista de espera**.

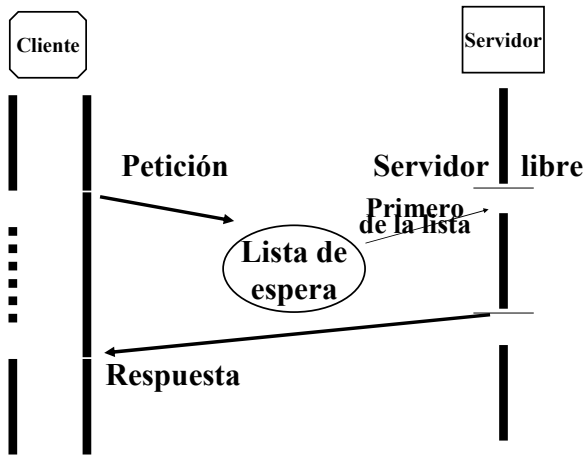


Figura 83. Servicio de Multicliente.

Independientemente de que el mecanismo sea síncrono o asíncrono, el cliente lanza la petición que se anota en la lista. Cuando el servidor queda libre se le suministra uno de las peticiones pendientes, normalmente la primera o la más prioritaria.

Si el modelo de comunicación es síncrono, la respuesta se envía directamente al cliente que la está esperando. Si es asíncrono, se anota en la lista a la espera de la recogida del cliente.

Finalmente, citemos un modelo que trabajaremos más adelante, Multicliente y Multiservidor, que se muestra en la figura y que permite que varios instancias del mismo servicio trabajen simultáneamente.

El cliente ignora que instancia es la que le da servicio. Este modelo permite una alta escalabilidad, pero lógicamente, aporta problemas de paralelismo que habrá que resolver.

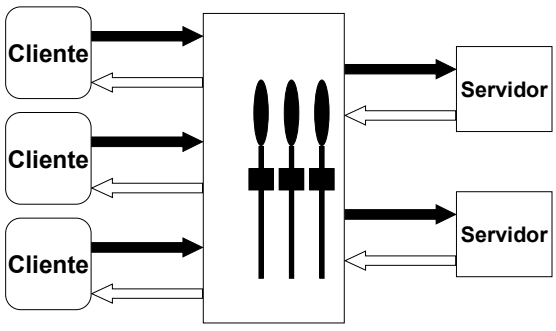


Figura 84. Modelo Multicliente y Multiservidor

Las opciones de hoy día, disponibles en compiladores a través de los mecanismos de multihilo, y Middleware permiten

montarlo y usarlo con esfuerzos razonables; si se necesita, claro. Naturalmente, hay otra limitación: la formación de los diseñadores.

5. Modelos de comunicación.

Los modelos de comunicación los veremos en profundidad en la segunda parte.

Para abrir boca, repasemos aquí alguno de los modelos incluyendo alguno de los tradicionales. Y aprovechemos para explicar la situación actual de estos últimos

Los símbolos que encontrará ligado a cada modelo es el icono que usaremos en la parte de desarrollo. Vaya familiarizándose con ellos.

5. 1. Modelos asíncronos.

5.1.1. Conversacional.

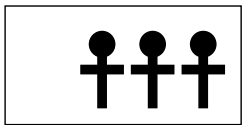
Voy a respetar la tradición. Pero no he entendido nunca la razón por la cual se ha de considerar el modelo conversacional como asíncrono dentro del mundo Cliente/Servidor, donde su uso ha sido siempre síncrono para reservar un recurso de forma que el cliente sepa en que momento le pertenece.

Cliente y servidor se enlazan directamente entre sí y se sincronizan mediante mensajes.

Supone la existencia de servidores dedicados y la reserva exclusiva del servidor.

Es un modelo C/S “contra natura” ya que supone reserva del servidor, situación, en principio no permitida en un diseño C/S.

5.1.2. Cola.



Vd. ya sabe perfectamente que es una cola. En el mundo de la comunicación C/S los clientes colocan mensajes en la cola y los servidores las cogen y procesan desde allí. Hay varios modelos de comunicación basados en colas que veremos en la segunda parte.

Es el modelo asíncrono por antonomasia. Su importancia es tal que le dedicaremos mucho tiempo después.

El ***Message Oriented Middleware (MOM)*** proporciona API’s de alto nivel para la gestión de colas con independencia de la plataforma de conectividad.

Sin embargo, en la segunda parte platearemos una polémica sobre ventajas e inconvenientes de utilizarlo o trabajar con un sistema de colas propietario.

5. 2. Modelos síncronos.

5.2.1. Entrada / salida remota.

Es un mecanismo desaparecido que permitía ejecutar una acción de e/s directa sobre un dispositivo remoto. Hoy día los dispositivos remotos de interés general son recursos compartidos a través del Middleware.

5.2.2. Servicio remoto.

Un programa pide un servicio remoto especializado sin conocer su localización dentro de la plataforma.

Hoy día son servicios remotos ampliamente utilizados:

- ODBC y ADO - Síncrono.
- OLE y Active X - Síncrono y asíncrono.
- Procedimientos catalogados - Síncrono.
- Servicios WEB. Petición de un servicio de WEB del que ya hemos hablado.

5.2.3. Petición de servicio remoto (RPC - Remote Procedure Call).

Es un mecanismo general para pedir de forma síncrona un servicio remoto.

Un cliente pide un servicio a un servidor y se suspende mientras no llega la respuesta. Es, pues una comunicación claramente asíncrona de tipología transaccional. Los parámetros se pasan como a una rutina normal.

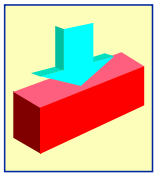
Es el mecanismo síncrono por antonomasia y el primer modelo de comunicación C/S que existió.

RPC hace la vida muy cómoda al cliente y al diseñador ya que permite delegar en el Middleware:

- La localización de los servidores.
- El paso de parámetros remoto.
- Como se han de tratar los fallos de conexión.
- Cual es el formato de los datos.
- La seguridad, encriptación, autentificación, etc.

Hoy día es un mecanismo estándar incorporado en el Middleware y que ha tenido un cierto auge debido a que TCP/IP, que se ha extendido como plataforma de comunicaciones, lo soporta de forma estándar.

6. Comunicación por eventos.



La comunicación por eventos es un modelo emergente que para que Vd. tenga una visión general de los modelos de comunicación, no puede dejar de citarse aquí.

Un evento supone la notificación entre cliente y servidor en el

momento necesario. Cuando el cliente envía una petición al servidor le puede enviar un evento y cuando el servidor devuelve la respuesta también pude hacerlo por un evento. Un error también puede notificarse por evento, etc.

Se suele decir que un evento es una forma de comunicarse asíncrona. Yo pienso que un evento no es ni síncrono ni asíncrono, es otra cosa, que permite, según la necesidad, trabajar síncrona o asíncronamente.

La comunicación por eventos es la forma natural de comunicarse entre objetos distribuidos.

Para que funcione correctamente el Middleware debe de disponer de un **Gestor de Eventos** que llegue a toda la plataforma de forma transparente. Y eso solo se tiene hoy día con un modelo de objetos distribuidos.

7. Comparación entre RPC y colas.

La comparación entre RPC y colas ha sido la forma de siempre para comparar las ventajas e inconvenientes entre modelos de comunicación síncronos y asíncronos.

Trataré en tema en profundidad en la segunda parte. Conviene, sin embargo, hacer una primera relación de ventajas e inconvenientes de cada uno de los modelos.

7. 1. RPC.

7.1.1. Ventajas.

- Facilidad de programación.
- Independiente de la plataforma lo que permite conexión con "sistemas desconocidos". Esconde las incompatibilidades de datos sobre plataformas y lenguajes.
- Aumento sin problemas de clientes.

7.1.2. Desventajas.

- Comunicación asíncrona.
- Hace más difícil un rendimiento escalonado.

7. 2. Colas.

7.2.1. Ventajas.

- Comunicación asíncrona. Si interesa síncrona se puede montar también.
- Versatilidad y rendimiento escalonado.
- Posibilidad de establecer prioridades y clientes especiales.

7.2.2. Desventajas.

- Mayor dificultad de diseño, programación y, en menor medida, de configuración.
- API's diferentes no interoperables en entonos no homogéneos.

Servicios y Modelos de Middleware.

1. Introducción.

El Middleware es el elemento clave de la integración del sistema distribuido. Además de aportar todos los elementos para conseguir transparencia en el diseño y la administración del sistema, incluye el transportista de forma tan estándar que ha conseguido que nadie se acuerde ya de su existencia.

Repasar los diferentes modelos de Middleware no tiene demasiado interés en si mismo. Es más, si no hubiera esta primera parte dedicada a la introducción de las arquitecturas distribuidas, no lo hubiera incluido.

Por esa razón el viaje va a ser rápido y sólo nos pararemos en un modelo genérico que como mínimo le aportará una idea de como trabajan los diferentes modelos. Y si desea ganar tiempo, vaya directamente al modelo genérico.

2. ¿Se acuerda del Middleware?

Antes de atacar sus modelos, recordemos que Middleware es la herramienta para conseguir transparencia eliminado la complejidad del sistema en:

- El desarrollo.
  - Obtener servicios de datos y de proceso de forma transparente.
  - Obtener transparencia en el transportista
- La administración del sistema.
- La localización de los elementos en la plataforma.

En la situación ideal el sistema y el transportista habrían de ser observados a través del Middleware como un **conjunto de API's que permiten acceder a estos elementos con la idea de sistema único**.

Las API's habrían de tener:

- Una **especificación única** (sintaxis). para cada servicio: precondición, parámetros y poscondición.
- Una **tipificación única** (semántica única). Cada clase de servidor, para la misma API, debería comportarse igual y dar el servicio de la misma forma.

3. La visión de sistema del diseñador a través del Middleware.

En la práctica, la visión del sistema que tiene el diseñador es la de la figura. Construye los programas cliente apoyando su lógica de presentación sobre las lógicas de datos y del proceso que a su vez utilizan el sistema a través del Middleware.

Escondido dentro del Middleware, el transportista cubre su función de localización servidores y transporte de peticiones de servicio.

Lo único que al final ve el programa cliente del sistema son las API's del Middleware.

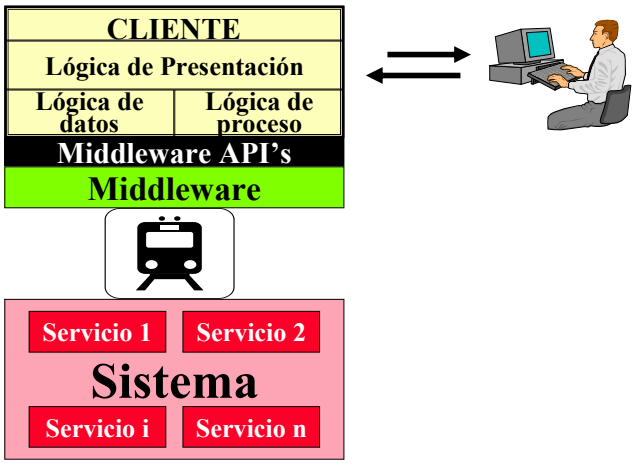


Figura 85. La visión del sistema del diseñador

Por favor, en esta visión idílica del sistema único, no olvide que el sistema puede fallar y que debe incluir el diseño de consistencia.

4. Componentes básicos del Middleware.

Los mecanismos básicos del Middleware, que deberá encontrar en todos los modelos, son:

- API's de alto nivel para la petición de forma transparente de los servicios.
- Un mecanismo de implementación e integración de los servicios construidos en el Middleware.
- Herramientas para referenciar, catalogar, gestionar y localizar los recursos en la plataforma.
- Facilidades de gestión distribuida.
- La plataforma del transportista.
- La interfície con las API's de bajo nivel del sistema.
- Si estamos en un modelo de objetos distribuidos, los recursos de gestión de estos objetos OO.

5. Servicios básicos proporcionados por el Middleware.

Incluyo a continuación una enumeración y separación de los servicios de Middleware entre diseño y administración. Notará sin embargo que hay servicios de interés en los dos grupos.

La lista no tiene el espíritu de ser exhaustiva. Sólo pretende mostrar que se puede esperar del Middleware.

Se incluyen también en algunos casos la terminología inglesa ya que es de frecuente utilización en este ámbito.

5. 1. Servicios de Desarrollo.

5.1.1. Acceso a recursos.

- Presentación.
- Impresión.
- Datos.
- Comunicaciones.

5.1.2. Servicios de comunicación entre programas.

- RPC.
- Colas (MOM).
- Conversacional.
- Memoria compartida (casi no se usa ya).

5.1.3. Servicios de Distribución.

- Referencia y catalogación de recursos y componentes.
- Arranque de recursos.
- Localización.
- Gestor de transacciones.
- Fecha y Hora.

5.1.4. Gestor de Objetos Distribuidos (Object Request Brokers ORB's)

Si el Middleware tiene un modelo de objetos distribuidos.

5.1.5. Servicios de administración del sistema.

- Servicios de referencia y catalogación de recursos (Naming and Directory Services).
- Seguridad, autentificación de usuarios y protección de acceso a recursos (Security Services).
- Algoritmos y protocolos de comunicación.
- Direcciones alternativas de transporte de mensajes (Alternative Message Routing).
- Gestión dinámica de recursos (Dynamic Resource Management).
- Gestor de Eventos (Event Management).
- Distribución de carga de trabajo (Load Balancing Services).
- Algoritmos de conversión automática de datos (Data Conversion Services) que comportan trasparencia de datos entre entornos heterogéneos.
- Integración transparente de protocolos diferentes (Context Bridging Protocols Services).
- Fecha y hora unificadas para todo el sistema distribuido (Global Date and Time Services).
- Inventario permanente de elementos.
- Servicios de Back-up y Restore.
- Registro de configuraciones.
- Métricas y programas para el análisis de rendimientos (Perfomance Services).
- Control y distribución de software.

6. Modelos de Middleware.

El primer modelo de Middleware que personalmente he visto ha sido el de DeBower&Dolgicer del año 1992.

El modelo está claramente pensado para proponer un Middleware que hiciera transparente los protocolos de red que hacían del transportista algo no estándar y, sencillamente, mortal de gestionar en aquella época.

El mismo año King proponía un modelo mucho más general y cercano a lo que uno espera hoy día del Middleware.

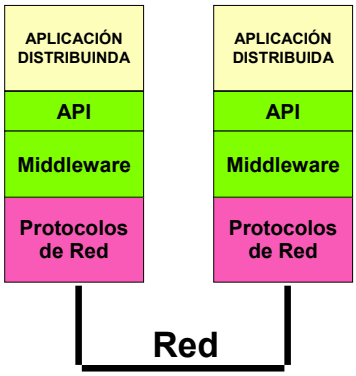


Figura 86. Modelo de DeBower&Dolgicer (1992)

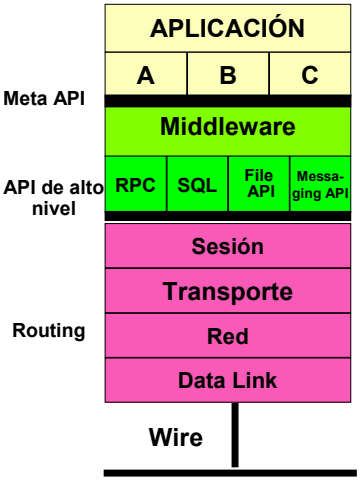


Figura 87. Modelo de King (1992).

Proponía una capa de servicios de alto nivel con RPC, SQL y MOM que se apoyaba en el concepto de sesión (se pide un enlace con el servicio, se usa y después se cierra). Las sesiones se apoyaban sobre una capa de transporte que hacia transparente la red y los datos.

En 1994 el mismo Dolgicer, y ya en solitario, proponía el modelo de la figura. Entre los servicios de alto nivel (RPC, DATABASE Access, MOM, ORB) y las aplicaciones se incluyen dos piezas llamadas a tener en el futuro una importancia básica: un gestor de transacciones y, sobre todo, el Mail, correo independiente de la heterogeneidad de la plataforma. El transportista queda definitivamente escondido en el Middleware y Dolgicer habla ya únicamente de un Middleware que transporta mensajes de forma transparente a la plataforma. Seguro que hay otros modelos anteriores que yo no conozco, pero por lo que sé, es éste el primer modelo de Middleware "completo".

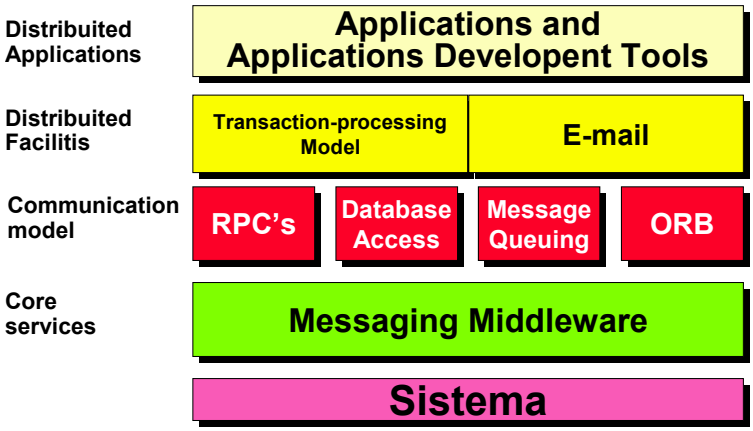


Figura 88. Modelo de Dolgicer de 1994.

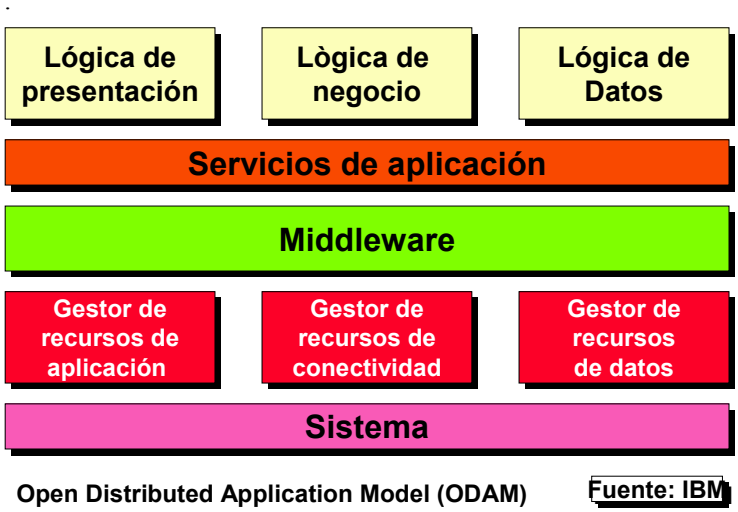


Figura 89. Open Distributed Application Model (ODAM).

Ya en 1996 IBM propuso el Open Distributed Application Model (ODAM) que se muestra en la figura. Está ante otro clásico que apareció durante años en todos los artículos de C/S dedicados al diseño. Observe, sin embargo, la gran similitud entre este modelo y el del 94 de Dolgicer. Su aportación es romper la aplicación en las tres lógicas, algo que aunque sea interesante, no aporta nada a un modelo de Middleware. Quizás su otra aportación interesante es crear una capa, los gestores, que se sitúa por debajo del Middleware, con una propuesta de servicios de DSM.

Otro modelo que no puede faltar en una relación de este tipo es el Windows Open Services Architecture (WOSA) propuesto y utilizado por Microsoft desde las primeras versiones de Windows.

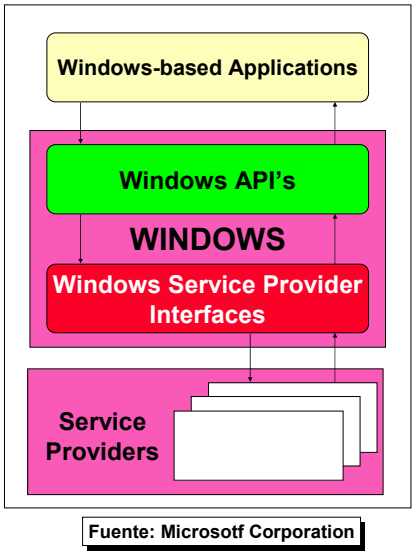


Figura 90. Windows Open Services Architecture (WOSA)

No puede faltar por su importancia dentro de la historia de la informática, no por su aportación técnica. Compare este modelo con el anterior. Es mucho más simple; y era para los productos Microsoft. Sin embargo, la buena gestión comercial de Microsoft llevó a la compañía a su posición de privilegio y todos los fabricantes de Middleware lo adaptaron.

El esquema es simple. Mediante API's proporcionadas por Windows, se ataca un sistema estándar de "empalme de servicios". Cualquier producto de Middleware que dé esa interfície, puede usarse desde Windows. Se pierde la organización de los servicios por capas proporcionados por los modelos anteriormente citados.

Una utilización de este modelo fue crucial es el desarrollo de aplicaciones distribuidas. Para poder conectar a cualquier motor de base de datos, Microsoft propuso una capa por debajo del Windows Service Provider Interface para enlazar con bases de datos relacionales. Se creó como una extensión de SQL, un estándar ya consolidado. Cualquier motor de bases de datos que proporcionara una conversión de sus API's nativas a esa capa podría ser atacado desde un programa Windows. Había nacido Open Database Connectivity (ODBC). ¡Aleluya!. El acceso trasparente a las bases de datos era una realidad. Y, por suerte, a pesar de una

resistencia inicial de los grandes proveedores de bases de datos, ODBC se convirtió en un estándar "de facto".

Un avance importantísimo fue la aparición de los modelos de Objetos Distribuidos que acabaron convergiendo en una propuesta estándar del Object Management Group (OMG): Common Request Broker Architecture (CORBA).

Microsoft reaccionó con COM y más tarde con DCOM. Más de lo mismo. DCOM y CORBA son el mismo en dos arquitecturas diferentes. Microsoft y los demás.

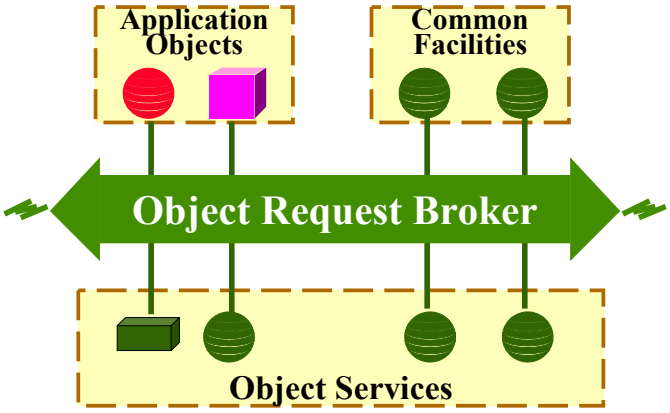


Figura 91. Common Request Broker Architecture (CORBA)

La presencia de dos arquitecturas diferentes y la necesidad de saber OO para trabajar con estos modelos de Middleware a frenado su popularización. Existe también una extensión de CORBA para Internet.

Considero que la importancia de un modelo de objetos distribuidos como Middleware es tal que se merece una apartado especifico. Presentaré CORBA (por aquello de que es la propuesta estándar.

7. Common Request Broker Architecture (CORBA).

CORBA define y proporciona:

7. 1. Servicios.

- Mecanismos de distribución de objetos.
- Servicios de DSM para la gestión de esos Objetos:
  - Manejo de clases.
  - Mecanismos de instanciación.
  - Servicios de localización y sincronización.
  - Servicios de catalogación, repositorio y directorio.
  - Integridad.
  - Query.
  - Versiones.
  - Acceso a Datos, etc. ..

7. 2. Object Management Architecture (OMA).

OMA soporta entornos heterogéneos y distribuidos combinando proceso distribuido y orientado a objetos. Proporciona una método estándar para crear, preservar, localizar y comunicar objetos distribuidos por la plataforma.

OMA puede ser dividido en cuatro componentes:

- **Objetos de Aplicación** (Application Objects), los clientes, que proporcionan la lógica la aplicación.

- **Servicios de Objeto** (Object Services) que proporciona las funciones estándar que los objetos necesitan para existir (integridad, catalogación, instanciación, etc..)
- **Utilidades de DSM** (Common Facilities) que proporcionan las funciones de DSM, como servicios de acceso a las BD, servicios de impresión, seguridad, reporte de errores, gestor de eventos, etc.
- **El Object Request Broker**

7. 3. Object Request Broker (ORB).

Proporciona una infraestructura que permite que los objetos se comuniquen entre si independientemente de la plataforma y de las técnicas y lenguajes utilizados en la implementación de la aplicación.

ORB recibe una petición de servicio, localiza el objeto que lo da, pasa los parámetros y devuelve los resultados. Todo ello según un modelo OO y de forma transparente a la plataforma.

ORB está preparado para dirigir:

- Servicios de referenciación (Name Services).
- Gestión de peticiones de servicio (Request Dispatch).
- Codificación de parámetros (Parameter Encoding)
- Sincronización (Synchronization).
- Servicios de arranque (Activación Services).
- Gestor de errores (Exception Handling).
- Mecanismos de seguridad (Security Mechanismes).

La estructura del ORB se esquematiza en la figura.

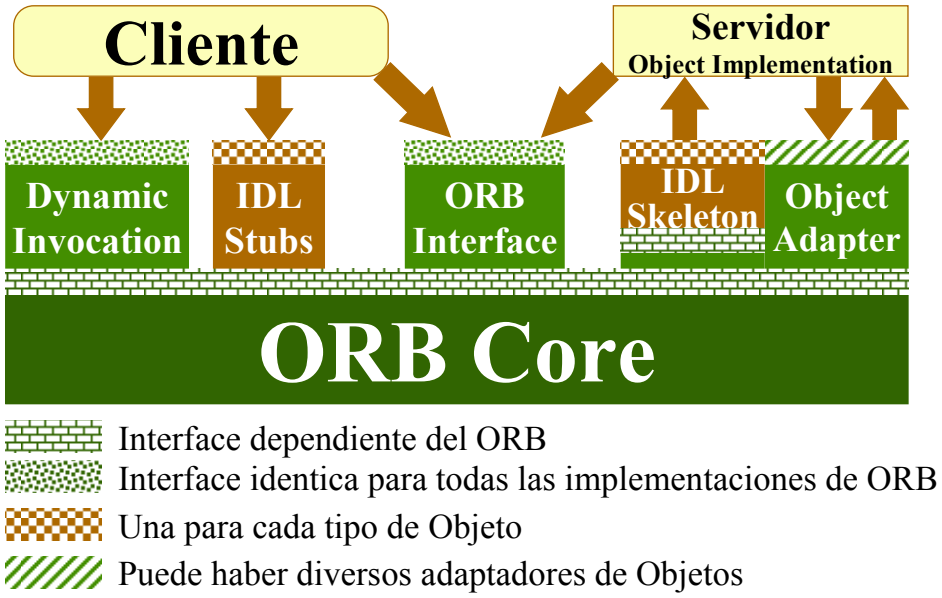


Figura 92. Estructura del ORB

Para hacer una petición de servicio el cliente utilizará la interfície proporcionada por el Dynamic Invocation y el IDL Stub del servidor (objeto) al cual pide el servicio. Para algunas funciones utilizará directamente el ORB Interface.

El servidor, implementado en un objeto, recibe la petición a través de una llamada generada desde el IDL Skeleton. Mientras procesa la petición, el Servidor puede utilizar el ORB Interface o el Object Adapter.

Como funciona el mecanismo de llamada y transporte de los parámetros utilizando los Stubs está explicado en la segunda parte de este libro cuando se explica el mecanismo RPC independientemente de la plataforma. No voy a repetirlo aquí. Si tiene interés avance una hojas y léalo.

La definición de los interfaces de los objetos puede realizarse de dos maneras:

- **Estáticamente**, a través de un lenguaje de definición de interfaces denominado Interface Definition Language (IDL). Los objetos pueden implementarse en cualquier lenguaje de programación y los Stubs y Skeletons definidos con el IDL permiten la intercomunicación transparente. Este hecho es fundamental. La panacea. ¡Programe en el lenguaje que quiera!. ¡Y después úselo de forma transparente!
- **Dinámicamente**, aprovechando un servicio del Interface Repository que permite añadir interfaces al formato de objeto y que pueden ser instanciados en tiempo de ejecución.

Estoy tan enamorado del los Objetos Distribuidos que tengo que hacer un esfuerzo para no seguir... Si desea más información consulte bibliografía especializada.

8. Modelo de Servicio WEB.

Recuérdelo aquí.

9. Un modelo estándar de Middleware.

La revisión de los modelos históricos de Middleware, aunque muy interesante y formativa tiene un interés relativo cuando se ponga a diseñar. Vd. tendrá servicios que usará como le convenga.

Por esa razón, lo que si es muy interesante desde el punto de vista de un diseñador es establecer un modelo general y estándar del Middleware independientemente del modelo concreto con el que trabaje. Diseñará con esa idea. Con la relación clara de los servicios de que dispone, sólo cuando implemente consultará la sintaxis concreta de cada servicio. Y delegará en el Middleware el trabajo. Como se lo monta el Middleware, es su problema, no el suyo.

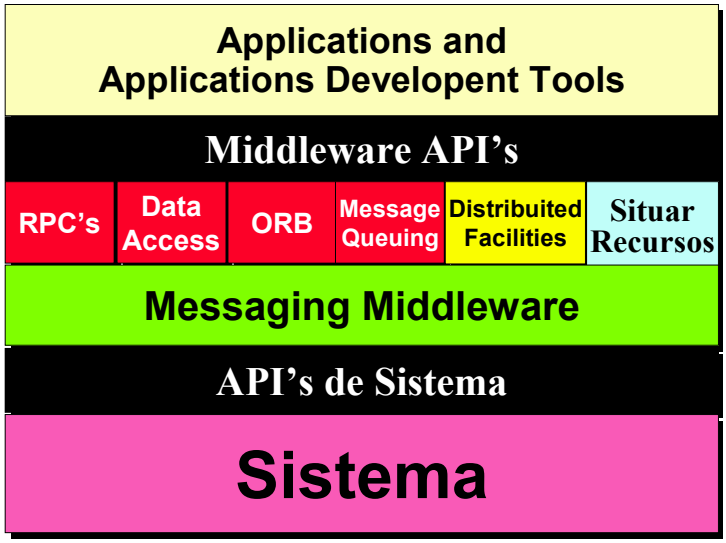


Figura 93. Modelo estándar de Middleware

Dentro de esa idea de establecer un modelo estándar de Middleware, le propongo que repase todos los modelos. Verá fácilmente, que desde el punto de vista esquematizado, todos tienen los componentes lógicos del modelo estándar que le propongo en la figura.

Las aplicaciones generadas a través de las herramientas de desarrollo usan todos los servicios a través de API's de Middleware. De hecho, esa es su única interacción con el Middleware.

A través de las API's usa dos tipos de servicios.

- Una capa de alto nivel: RPC's, Data Access, ORB, MOM, etc.
- Las Distributed Facilities proporcionadas por el DSM. En particular, son fundamentales las de situación de recursos (instanciación, directorio, localización, etc.).

Todos los servicios se apoyan en una plataforma de traspaso de mensajes (Messaging Middleware) que permite comunicar de forma transparente clientes y servidores. Esta plataforma es la que utiliza las API's especificas del sistema de cada plataforma. Pero a un nivel de profundidad que el diseñador apenas intuye.

La idea del Middleware estándar se completa con un ideal: una semántica y una sintaxis únicas para cada servicio.

Todo el conjunto operativo tiene un aspecto lógico como el que se muestra en la figura que podría representar un esquema de un sistema distribuido específico. Observe la presencia de la posibilidad, hoy día lo normal, de clientes y servidores arrancados en el mismo servidor físico.

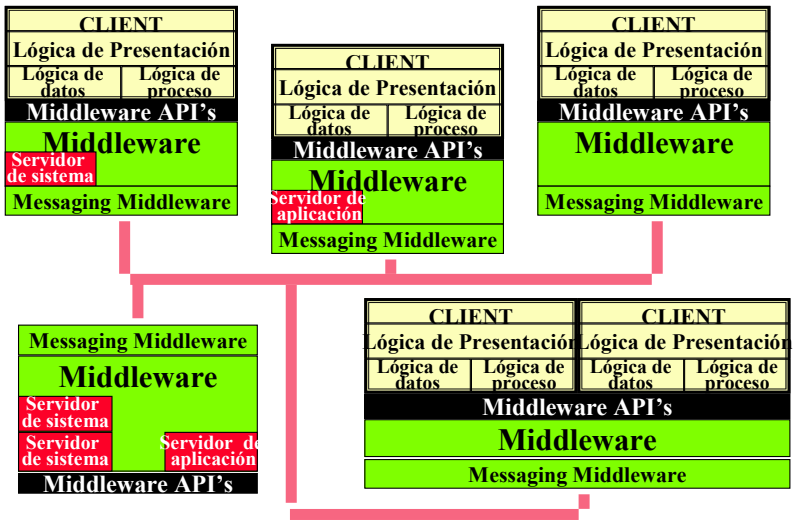


Figura 94. Una distribución de Middleware en un sistema distribuido

10. El modelo real de Middleware.

La situación real no es tan idílica:

- Coexisten varios productos para una única línea de servicios.
- Existe Middleware de usuario desarrollado para cubrir aquellas funciones que no proporciona el Middleware estándar. Puede ser de dos tipos:
  - Middleware totalmente aportado por la aplicación..
  - Middleware desarrollado para encapsular heterogeneidad de diversos Middleware con los mismos, parecidos o equivalentes servicios aunque con sintaxis diferentes.

Todo ello comporta que el modelo estándar del Middleware debe completarse con estos elementos para completar el Middleware real tal como se muestra de la figura.

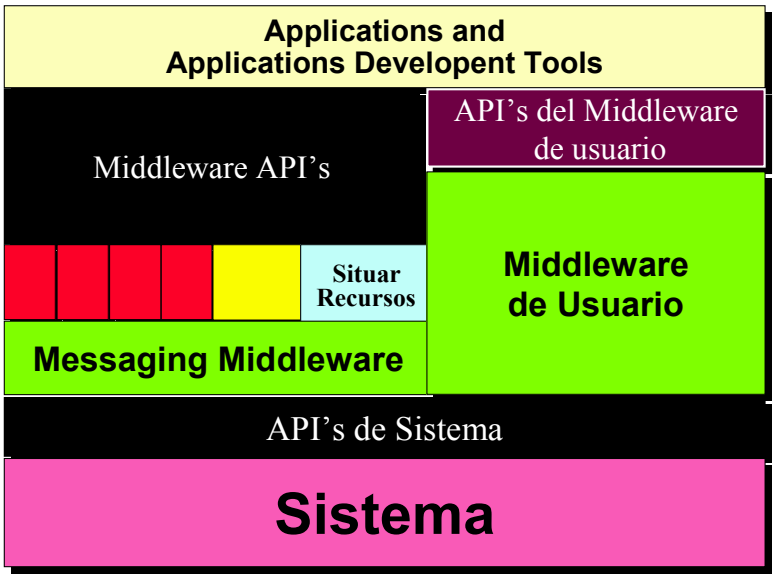


Figura 95. Modelo de Middleware Real

La evolución de los estándar en continuada. Con ella el Middleware desarrollado como de usuario puede ser cubierto por un nuevo estándar. La pregunta surge espontáneamente: ¿hay que sustituir el Middleware de la instalación por el estándar?

Yo soy un convencido de que sí. Al hacerlo adaptaremos automáticamente muestro diseño al carro de la evolución. Si no, cada vez que se de un paso adelante, el Middleware de usuario deberá adaptarse. Reconozco que muchas veces la sustitución tiene un componente personal y emotivo. Sustituimos un Middleware que nos ha costado un montón de horas de trabajo y del que nos sentimos orgullosos. Es como matar un hijo. Pero, a pesar de ello, hágalo.

La vía para hacerlo es fácil. Lo normal es que el contenido semántico y sintáctico del nuevo Middleware no coincida con el del usuario que va a sustituir.

Tiene dos caminos:

- Sustituir en bloque, cosa que muchas veces no es factible, no solo por la carga de trabajo, sino por la distribución de la plataforma.
- Encapsular el nuevo Middleware con llamadas con la sintaxis y la semántica antiguas y usar esta nueva implementación a través de una compilación masiva. Y a partir de ese momento utilizar el nuevo Middleware directamente en el software nuevo y en los programas que se modifiquen. Llegará un día en que lo que quede por sustituir será mínimo. Entonces haga el esfuerzo final y descatalogue el Middleware de usuario obsoleto.

Implementación e Integración.

1. Introducción.

El objetivo de diseño, plasmado en la implementación e integración del sistema, es conseguir sistemas distribuidos con independencia de la infraestructura y de la localización de los servicios.

Vamos a responder aquí a la pregunta de como conseguirlo. Sin embargo, dudo que Vd. necesite ya este capitulo. Si le he mostrado correctamente el concepto y posibilidades del Middleware, este capítulo le parecerá suficiente y trivial. Si no, le parecerá sorprendentemente corto.

2. Implementación.

La implementación del sistema se realizará utilizando a fondo los servicios del Middleware.

Si no dispone de algún servicio, intente comprarlo fuera.

Si aún así no lo consigue, constrúyalo como Middleware de usuario. Pero hágalo de forma que pueda encajarlo dentro de la zona de servicios especializados del esquema de Middleware estándar (al lado del RPC, MOM, etc...). Para construir este Middleware de usuario, utilice a fondo todos los servicios del Middleware estándar.

Y a medida que vayan saliendo estándares, sustituya su Middleware de usuario por Middleware estándar.

Si tiene un Middleware heterogéneo, encapsule uno de ellos con la semántica y la sintaxis del otro. Escoja como marco el más estándar.

3. Integración.

Catalogue sus componentes, clientes, servidores y ampliaciones de la plataforma, con los recursos del DSM de su Middleware. Utilice para ello las Distribuited Facilities del modelo estándar. Si lo hace así, sus programas, elementos de la plataforma y Middleware de usuario podrán ser tratados como cualquier otro elemento del Middleware estándar y tendrá un único marco de administración.

Se puede hablar de integración del sistema distribuido en dos sentidos:

3. 1. Desarrollo.

La relación entre las piezas puede establecerse:

3.1.1. Estáticamente.

En el momento de linkar la aplicación.

3.1.2. Dinámicamente

En fase de ejecución. Los clientes usarán a los servidores, estos a otros servidores, y los clientes se comunicarán por interfase.

3. 2. Explotación.

La integración funcional del sistema distribuido puede ser de dos tipos:

3.2.1. Estática.

Definida al arrancar el sistema a partir de la configuración definida con los recursos del DSM.

3.2.2. Dinámica.

Modificada y gestionada por los administradores del sistema distribuido en tiempo de ejecución y con los recursos del DSM.

Iniciando el camino

1. Introducción.

¿Qué ha de hacer una organización que quiera empezar a diseñar aplicaciones sobre arquitecturas distribuidas?

En este capítulo vamos a comentar algunos aspectos a tener en cuenta si la suya ha de empezar a recorrer ese camino.

2. La influencia del tipo de aplicación.

Hemos visto a lo largo de la primera parte que, de hecho, hay dos tipos de aplicaciones distribuidas: las de desarrollo rápido (RAD) y las avanzadas.

La situación es muy diferente su Vd. desea que su organización esté preparada para la primera o la segunda situación.

Si desea entrar con aplicaciones RAD el camino en sus primeros pasos es muy corto ya que su organización sólo debe tener en cuenta cuatro factores de todos los que vamos a exponer a continuación:

- El concepto de Middleware.
- Qué una aplicación distribuida, cualquiera que sea su tipología ha de prever que pasa cuando no responden los servidores, lo que hemos llamado análisis de consistencia.
- Qué el hecho de que la aplicación sea RAD no le exime de trabajar con la idea de componentes reutilizables.
- La interacción y la sinergia entre Cliente/Servidor e Internet.

A continuación le presento como pienso, en función de mi experiencia, que una organización debe prepararse para desarrollar aplicaciones distribuidas avanzadas. Si quiere prepararse únicamente para las de desarrollo rápido, seleccione del total lo que hace referencia a los factores arriba referenciados.

Analizaremos los primeros pasos en cuatro factores:

- Planificación.
- El factor humano.
- El primer proyecto.
- El ciclo introductorio.

3. Planificación.

La introducción de una organización en el desarrollo de aplicaciones distribuidas pasa por la necesidad de establecer una estrategia para hacerlo. Y esa estrategia debe concretarse en un plan. Establecer un plan es una necesidad. Omitirlo comporta casi inexorablemente el fracaso.

Debe analizarse la situación de la infraestructura informática de la compañía, decidir donde se quiere llegar y establecer un plan de evolución de la situación actual a la futura.

El **plan de evolución** ha de marcar los criterios y estándares. La infraestructura informática global ha de ir adaptándose al aumento de elementos interconectados. La evolución ha de ser gradual y sin traumas.

En particular, la plataforma de conectividad ha de estar definida claramente ya que redes y comunicaciones resultarán fundamentales.

La organización deberá decidir si desea ir todo el camino con sus propios recursos o se va a apoyar en expertos contratados por Outsourcing.

Deberá decidirse cual son los niveles lógicos que se establecerán sobre los niveles físicos y cual será **la política general de administración del sistema distribuido**, y, en particular, cual será la política de soporte a usuarios.

Todo ello quedará reflejado en el **Plan Estratégico de la Plataforma Distribuida** que servirá de referencia en el camino. Sin embargo, no existirán verdades eternas, la experiencia en el camino ira adaptando el modelo a la necesidad. Adapte, pero deje siempre los criterios de su organización perfectamente claros en el Plan.

4. El factor humano.

La introducción del paradigma de diseño distribuido en una organización con experiencia no es un tema complejo.

Las organizaciones de HOST habrán de superar el trauma que para los profesionales de este mundo supone el hecho de que el desarrollo distribuido es más complejo y menos dogmático que el centralizado. El profesional va a trabajar más. Y ha de estar mejor formado. Y si no ve las ventajas que su organización tendrá, sólo vera en la evolución un paso atrás que le complica su hasta entonces plácida y controlada vida.

Encontrará dos reacciones. El profesional que se enquista en la solidez del proceso centralizado y el profesional que encontrará en la llegada del nuevo paradigma una posibilidad de aumento de experiencia profesional. Muchas veces la diferencia entre una y otra actitud estará en la información y el nivel de estabilidad de su organización informática. Mi experiencia me dice que la edad no es, en general, el factor que discrimina una u otra actitud. Tiene mucha más importancia la formación. Y en dos sentidos:

- La previa de cada persona: cuanto mayor sea el nivel de formación de la persona más abierta estará a nuevas experiencias.
- La formación que se dé antes de iniciar el camino: si forma a su personal, estará más motivado y tendrá menos miedo al cambio.

No olvide, pues, que el personal afectado ha de ser **formado, motivado y entrenado**. La formación se consigue con cursos. La motivación informando, explicando las ventajas a conseguir y haciendo participes de las decisiones al equipo humano Y en entrenamiento desarrollando pequeñas partes de proyectos reales tutorizadas como última parte del proceso de formación.

Incorpore a esos proyectos tutelados personas externas con experiencia. Después, las personas de su organización que mejor se adapten pueden ser los tutores de otros grupos y de esa forma, la formación y la experiencia fluirán por toda su organización.

Si usa OutSourcing, no dude en traspasar los procesos antiguos al personal externo y hacer participar al interno en los nuevos.

Esta decisión no siempre es fácil ya que como la gente interna conoce mejor los procesos de toda la vida la calidad de servicio puede bajar. Controle que no sea por debajo de límites tolerables. Y recuerde que el futuro es lo nuevo y el pasado lo antiguo. Su gente le interesa en el futuro no en el pasado. Además de la motivación que conseguirá actuando con esa mentalidad.

5. El primer proyecto.

El primer proyecto ha de tomarse como de training y evaluación.

Una buena opción es escoger un pequeño proyecto o hacer reingeniería de una parte pequeña de un sistema ya existente.

Pueden hacerse recomendaciones en la elección del primer proyecto:

- **Escoja un proyecto real pero sin plazo de entrega critico**. Intente que no tenga condicionamientos de plataforma importantes o bien lo que necesite de ella ya esté operativo.
- El ámbito del proyecto ha de estar bien definido y no ser muy ambicioso.
- Escoja un equipo de gente preparada y motivada, dirigidos por un Jefe de Proyecto competente e interesando en las nuevas tecnologías.
- Consiga que el equipo se dedique exclusivamente al proyecto.
- Dos usuarios líderes, a ser posible:
  - Uno en un puesto alto del organigrama de la compañía.
  - Otro que se juegue su prestigio en el proyecto.

Una vez cumplida esta primera etapa, escoja una aplicación piloto. Los objetivos de esta aplicación piloto son:

- Calibrar el coste del cambio tecnológico dentro de la organización.
- Adquirir formación y experiencia en las tecnologías distribuidas.
- Evaluar las ventajas que puede tener para nuestra organización y en que ámbitos conviene aplicarlo.

Son criterios a aplicar en la selección de la primera aplicación piloto:

- Una definición funcional muy clara y a ser posible con valor añadido importante.
- Que afecte a áreas de la organización motivadas por el cambio.
- El proyecto debe de estar impulsado, una vez más, con un mecenas bien situado en el organigrama de su compañía.
- Debe estar perfectamente definida la evolución necesaria en la plataforma desde la situación inicial a la que necesita el proyecto. Y el cambio no ha de ser muy traumático. Si lo es, debe atacarse como un proyecto previo e

independiente. En particular, la infraestructura de conectividad debe definirse correctamente.

- ❍ Los productos de Middleware han de estar perfectamente definidos dentro del Plan Estratégico.
- ❍ La localización de los datos, una vez establecido el modelo distribuido, ha de ser clara: lugar, modelo, mantenimiento, volumen y **administración**.
- ❍ Debe prever concienzudamente el **diseño de consistencia**.

6. El ciclo introductorio.

Depende mucho de la organización y del tiempo que le puede dedicar el equipo inicial (si no tiene dedicación exclusiva).

En el siguiente cuadro le resumo un posible ciclo introductorio, para desarrollar proyectos distribuidos avanzados y con componentes reutilizables, en función de mis propias experiencias.

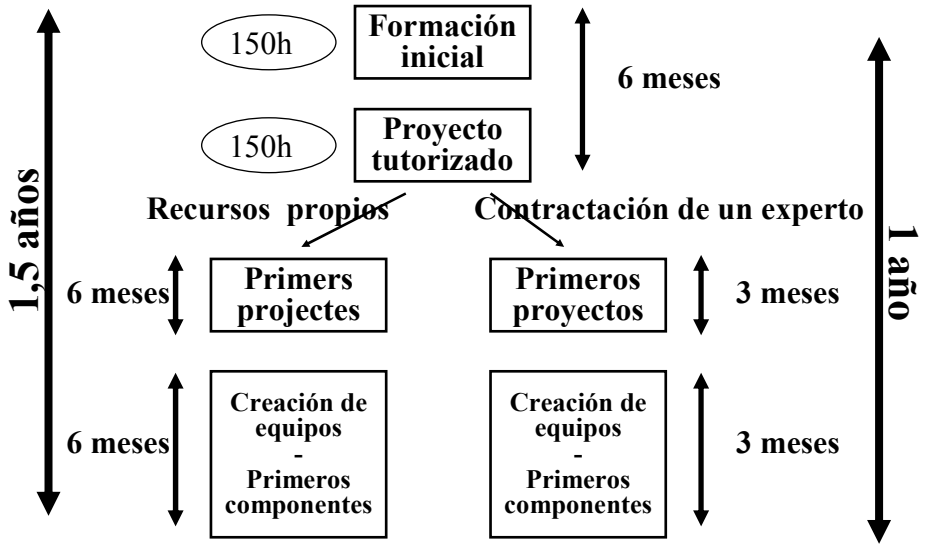


Figura 96. El ciclo Introductorio

Obviamente los plazos están en función de los recursos y la dedicación asignados al proyecto y de la carga de trabajo de los proyectos elegidos. Tómese los valores de la figura superior con todas las precauciones del mundo y orientativamente.

7. Consideraciones subjetivas.

- ❍ Hay dinamizadores imparables:
  - ❍ **Las empresas necesitan adaptación rápida de la informática a sus cambiantes procesos de negocio**.
  - ❍ El puesto de trabajo del usuario es hoy día el origen de la iniciativa informática.

- El protagonismo de la Ofimática ha conseguido máquinas clientes muy potentes y desaprovechadas.
- La presión de los menores costes teóricos (sin embargo vea más adelante el capítulo de tópicos).
- La presión del ambiente.

- La adaptabilidad de los sistemas distribuidos.
- El fácil acceso a redes y comunicaciones.
- Se ha de confiar en el Middleware.
- El cambio de mentalidad no es fácil en organizaciones de diseño centralizado.
- No todo puede pasarse todo a distribuido. HOST, C/S sobre Sistema Operativo e Internet coexistirán y se reforzarán mutuamente.

ERP's

1. Introducción.

Hoy día, una forma de montar sistemas distribuidos es utilizar ERP's.

El concepto conlleva aplicaciones corporativas, parametrización para conseguir la especialización y aplicaciones periféricas en una clara arquitectura cliente/servidor bajo presentación gráfica tipo Windows y/o Internet.

Hacer una visita rápida a este tema, es interesante no sólo para conocer este tipo de productos, una clara posibilidad para distribuir, sino también porque son ejemplo de sistemas distribuidos.

2. ¿Qué es un ERP?

Un **ERP** (**Enterpise Resource Planning Information System**) es un producto de Software integrado, compuesto por un conjunto de módulos prefabricados, que dan una solución estándar a los principales sistemas de información de una empresa:

- Contabilidad, Finanzas y Tesorería.
- Gestión comercial
- Facturación.
- Producción.
- Recursos Humanos: nómina, control de presencia, gestión de recursos, motivación, etc.
- Relaciones con clientes (CRM – Client Relation Management).
- Cadena de suministros (SCM – Supply Chain Management).
- Gestión de canales indirectos (PRM – Partner Relationship Management)
- Estadísticas de análisis de la Gestión, etc..

Los módulos son desarrollados e integrados por el fabricante y proporcionan varias herramientas adicionales fundamentales:

- Una amplia posibilidad de parametrización.
- Un lenguaje específico de programación.
- Una herramienta de generación de informes (**reporting**).
- Exportación de datos a MS Office.
- Posibilidad de utilizar desde un lenguaje de programación convencional objetos con la semántica de los datos y servicios del paquete. Se concreta en una librería de funciones.
- Herramientas de presentación gráfica y personificación de menús.
- Herramientas de test.
- Herramientas de Workflow.
- Conexión EDI y XML.
- Conexión con mail.

Es decir, el ERP proporciona un entorno propio de desarrollo con el cual el proveedor también ha diseñado y construido todos los módulos estándar entregados al cliente.

El cliente puede desarrollar funcionalidad adicional exactamente igual que lo haría el proveedor e integrarla con la básica en una arquitectura única.

El ERP promociona también herramientas de administración, autentificación y monitorización.

Todo ello, debidamente utilizado, permite generar una solución específica para cada caso y abrir la accesibilidad a la información desde toda la organización con muy poco esfuerzo técnico.

3. Ciclo de vida de un ERP.

Como todo sistema de información, un ERP también tiene un ciclo de vida.

3. 1. Etapa de reflexión.

La compañía debe valorar si le conviene la adopción de un ERP. En la decisión deben implicarse directivos, usuarios e informáticos.

En esta etapa deben utilizarse consultores externos, ya que es la única forma de saber que proporcionan los ERP's y como lo hacen. No debe permitirse, sin embargo, que esos consultores tomen la decisión, ya que en ese caso, sáltese está etapa; como es evidente, a ellos les interesará que se decida por un ERP.

3. 2. Etapa de pre-arquitectura.

Si la decisión ha sido ERP, le recomiendo que adelante la etapa de estudio de cómo será la arquitectura de los sistemas de información de la compañía bajo en prisma del ERP:

- Que módulos del estándar necesita adoptar.
- Que nivel de personalización necesita, que módulos habrá que programar a medida.
- Que condicionamientos organizativos está dispuesto o se puede permitir aceptar sin perder eficacia en los procesos de negocio.
- Que información, entre las inmensas posibilidades del ERP, realmente necesita.
- Que posibilidades nuevas le puede aportar el ERP, etc..

Esta etapa le va a dar elementos y argumentos para la siguiente de elección del ERP.

Es una etapa desagradecida ya que necesita tiempo para plantear, valorar y priorizar varios escenarios. Y la final, si el trabajo está bien hecho, solo hay propuestas concretas que parecen no justificar todo el tiempo empleado.

3. 3. Etapa de selección.

Deberá evaluar y escoger, entre la oferta que se le presente, el ERP que más le convenga.

Los criterios se pueden agrupar en grupos:

3. 4. Criterios de negocio.

- Estratégicos, que variaran en cada caso
- Coste versus ahorro por mejoras y eliminación de costes internos.
- Definir el alcance del proyecto.
- Que puede aportar el ERP a su negocio con sus módulos ya construidos y que ahora usted no tiene.

3. 5. Criterios de funcionalidad.

- Nivel de adaptación del ERP a su negocio y al revés.
- Presencia de adaptaciones verticales del ERP que ya tengan parte de la personificación, que necesitan sus procesos de negocio, ya realizada.
- Mejoras organizativas que le permitirá el ERP.

Como ve, en esta lista no está un clásico: la adaptabilidad del ERP a su negocio; hablamos solo de nivel de adaptación. Seguro que cualquiera ERP se adaptará a su negocio si Vd. pone un mínimo por su parte.

3. 6. Criterios técnicos.

- Plataformas soportadas. Aquí, si la solución que necesita es importante en recursos de plataforma, valore LINUX como sistema operativo para la base de datos y los servidores de comunicaciones.
- Gestores de bases de datos soportados.
- Posibilidades y rendimiento de las comunicaciones si el negocio tiene extensión geográfica.
- Facilidades y posibilidades de las herramientas de personificación y desarrollo. Valore aquí la oferta de profesionales del mercado en esas herramientas: cuanto más extensa sea, más barato le costará el sistema y más seguridad de continuidad tendrá.
- Documentación técnica y de usuario disponible.
- Gestión de la seguridad.
- Si su negocio está implementado en varios países, multiidioma y adaptación a la fiscalidad y reglas oficiales de las diferentes naciones.
- Facilidad del transitorio desde los sistemas actuales al ERP.
- Seguimiento de los estándares.

Observe que no está la posibilidad de conectividad con otros paquetes y ofimática: todos los ERP's actuales tienen soluciones más que suficientes en este apartado

3. 7. Criterios de consultaría.

- Servicios disponibles en el mercado.
- Número de consultarías que los ofrecen.
- Metodologías de implementación que aportan.
- Tantee costes (y asustese).

3. 8. Criterios de proveedor.

- Solvencia: empleados, facturación, recursos de desarrollo, clientes y entre estos los que se dedican a lo mismo, localización del soporte, etc.

- Si su negocio está implementado en varios países, disponibilidad de soporte en ellos.

3. 9. Criterios de implantación.

- Costes de:
  - La personificación.
  - Licencias.
  - Mantenimiento anual.
  - Personal interno necesario.
  - Directo (cursos) e indirecto (tiempo del personal interno) de la formación.
- Tiempo de necesario para la implantación.

Todo ello debe concretarse en:

- Un presupuesto de la inversión inicial y del gasto anual.
- Un contrato con la consultaría y/o el fabricante.

3. 10. Criterios de pre-viabilidad.

Que su negocio esté al nivel del paquete y al revés.

Por favor, reflexione sobre lo que su compañía puede permitirse pagar el coste necesario para hacer bien las cosas, de inversión inicial y de gasto anual. **No sea que**, dicho en palabras muy duras, **no se lo pueda permitir.**

3. 11. Etapa de compra.

Donde se concretarán los contratos. Haga participar a sus abogados.

3. 12. Etapa de implantación.

Necesitará consultoría externa para atacar las subetapas en las que realizará la implantación:

- Análisis de la adaptación entre el ERP y la compañía.
- Programación de las personalizaciones del ERP.
- Programación de las aplicaciones periféricas.
- Parametrización.
- Migración desde los sistemas informáticos anteriores.
- Formación de los usuarios.
- Arranque.
- Evaluación del arranque.
- Adaptaciones post-arranque.

No olvide que aquí la adaptación es doble: del ERP al negocio y del negocio al ERP.

La gestión del proyecto en la implementación del ERP es fundamental. Será el mejor antídoto contra los problemas capitales de la implantación:

- Tamaño y tiempo del proyecto.
- Rotación del personal.
- Resistencias internas.

- Gestión del riesgo.
  - Puntos de fallo en la planificación.
  - Puntos organizativos de resistencia.
  - Coste y posibilidad de paralelos con los sistemas en caso de problemas.
- Planificación optimista, uno de los pecados más frecuentes y graves.
- Levantar expectativas excesivas.
- Interfaces no previstos.

Estos problemas se resumen en dos:

- La inversión prevista se dispara.
- Retrasos y tensiones en la puesta en marcha

Una figura clave en esta etapa es el Jefe de Proyecto. No necesariamente ha de ser un experto en el ERP. Es mejor primar una persona flexible, metódica, con capacidad de tomar decisiones, hábil negociador, líder de equipos humanos y con imagen. Habrá de ganarse el respeto y la confianza de todos los actores del proyecto. Si saber negociar con ellos, tanto en fase de diseño como de resolución de problemas.

Al final de esta etapa **debe conseguir, como objetivo paralelo básico, que el know-how quede en personal interno y nunca únicamente en los consultores externos**.

Para ello, motive a su personal interno. Como hemos comentado antes, es mejor que subcontrate el día a día y permita a sus empleados dedicarse al nuevo proyecto.

Esto es muy fácil de decir y muy difícil de conseguir ya que:

- Los sistemas actuales deberán mantenerse operativos mientras no llegue el ERP.
- El personal actual es el único que tiene el know-how que permitirá la migración del sistema actual al ERP.

3. 13. Etapa de explotación.

Incluye el uso, administración, mantenimiento y evolución de la solución ERP con los procesos de negocio y la evolución técnica de la informática.

3. 14. Etapa de abandono.

Que no es sinónimo de fracaso. Puede ser que su empresa necesite cambiar de ERP o evolucionar a otro sistema de información según las estrategias de la compañía.

No hay nada eterno, ni siquiera un ERP.

4. Arquitectura de un ERP.

La arquitectura de los actuales ERP se estructura alrededor de los tres niveles habituales

- **Base de datos**
- **Aplicación**, la lógica de proceso.
- **Presentación**.

El componente de presentación puede estar orientado a:

- **Cliente / servidor sobre sistema operativo, en general Windows.**
- **Internet.**

En la figura se muestran las tres arquitecturas posibles de los ERP's modernos.

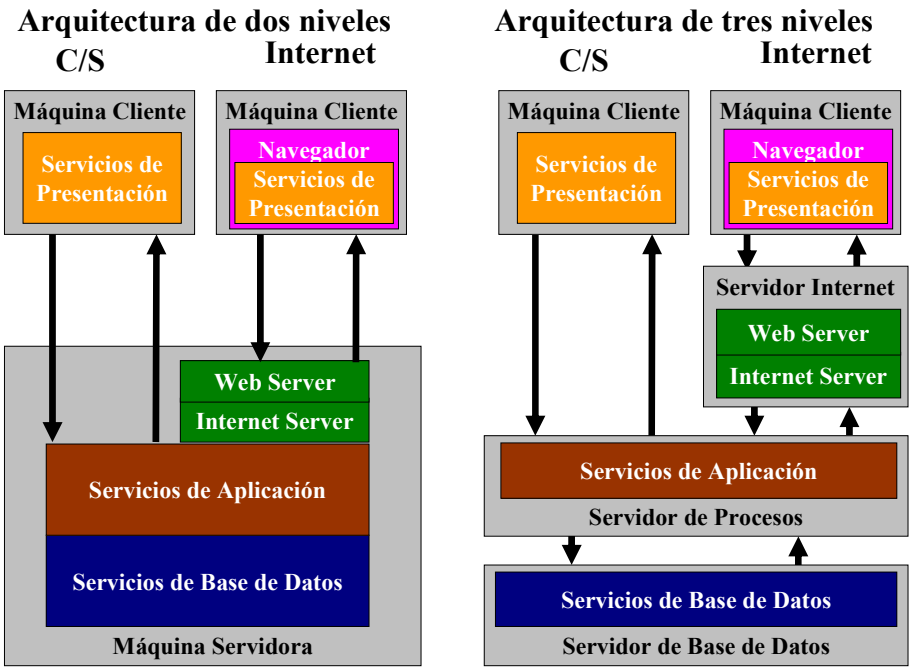


Figura 97. Arquitectura de un ERP

Los elementos involucrados en la arquitectura son:

- **Servicios de base de datos,** que proporcionan el almacén y el acceso a los datos.
- **Servicios de aplicación**, que proporciona, además de la lógica de proceso, el **Monitor de Transacciones** y la **Administración del Sistema**.
- **Internet Server**, que procesa las Transacciones de Internet.
- **Web Server** que maneja el Acceso a Internet.
- **Servicios de Presentación GUI en arquitectura C/S**.
- **Servicios de Presentación en arquitectura Internet**.
- El **Navegador** para gestionar los servicios de presentación de Internet, que obviamente, no forma parte del ERP..

Básicamente, estos componentes se organizan en tres arquitecturas:

4. 1. C/S de dos niveles.

Toda la carga de servicios, datos y aplicaciones, se concentra en una máquina servidora. El cliente solo tiene el componente de presentación, que

puede ser C/S sobre Sistema Operativo o Internet. En la inmensa mayoría de los casos, la arquitectura de dos niveles utiliza el modelo de presentación sobre Windows.

Esta arquitectura permite un ahorro de hardware. Puede utilizarse en implementaciones especiales, instalaciones pequeñas con un número reducido de usuarios (situación nada normal por el coste del ERP) y entornos de test.

4. 2. C/S de tres niveles.

Esta arquitectura permite balancear la carga entre máquinas cliente y servidor.

Hay un servidor de datos único y los servicios de aplicación se pueden colocar en máquinas dedicadas o en las máquinas cliente, lo que permite una gran escalabilidad. Esta última arquitectura conlleva clientes pesados o mayor inversión de hardware pero un mayor rendimiento en los procesos y menor riesgo a ruptura de servicio por averías de hardware.

Normalmente el concepto de tres niveles de los folletos suele referirse a máquinas servidoras diferentes para datos y procesos y no al hecho de haber servicios de lógica de datos individualizados.

El componente de presentación habitual es Internet.

4. 3. C/S de tres niveles con proceso paralelo.

También llamada en algunos casos, SAP entre ellos, arquitectura Multi-nivel.

Es un caso particular del anterior pensado para Internet. El Web Server concentra el flujo con los clientes Internet y se comunica con el Internet Server que enlaza los servicios del ERP con la plataforma Internet.

La arquitectura, que sigue siendo de tres niveles, permite poner tantos servidores en paralelo como necesidades de proceso generen los usuarios conectados. Un buen ejemplo, pues, de paralelismo.

5. La arquitectura de capas.

En el mundo de los ERP's es muy habitual hablar de capas: de un nivel, de dos; capa de presentación, capa de datos y de presentación, etc….

La terminología de rígida de capas en el mundo del diseño distribuido es un concepto superado hace tiempo. Hoy día planteamos el diseño distribuido como una arquitectura de servicios que se interrelacionan entre si para proporcionar la funcionalidad deseada. La arquitectura SOA no es más que eso.

La realidad es tan compleja que es imposible, **e inútil**, estratificar los servicios que se utilizan excepto en aplicaciones muy pequeñas o en productos, como los ERP's en que la arquitectura de capas es casi un condicionamiento de diseño, más por razones históricas que por limitaciones técnicas.

Es más, los mimos ERP's exitosos han evolucionando de tal forma que se han visto obligados a introducir conceptos como mullticapa para intentar mantener la terminología de capas.

No me parece un camino a seguir. Mi espíritu es de SOA, no de capas.

Sin embargo, es conveniente que un diseñador distribuido dentro de su cultura general conozca esta terminología. Y valgan estos productos como ejemplo.

5. 1. Modelo de Programación.

Los ERP's modernos están diseñados con el modelo avanzado orientado a objetos, basado en una arquitectura de objetos distribuidos.

Las modificaciones han de hacerse, pues sobre objetos. Hay dos formas de hacerlo:

5. 2. Modificando los objetos directamente.

Un grave error a mi juicio ya que cuando se cambia la versión de ERP hay que repetir toda la personalización. Ello convierte los cambios de versión en un proceso desagradable y caro.

5. 3. Utilizando técnicas de herencia y polimorfismo.

El camino recomendable. Sólo tiene un inconveniente: hay que saberlo hacer.

6. Ventajas de los ERP.

Las ventajas de los ERP's son fundamentalmente.

- Integración de todos los procesos básicos de la empresa en un único sistema de información.
- Unificación del modelo lógico de datos.
- Una vez arrancado y en explotación, flexibilización a los cambios organizativos y a la incorporación de nuevos módulos.
- Integración de compañías de grupos heterogéneos.
- Integración de arquitecturas centralizadas, cliente / servidor basado en SO e Internet, Workflow y la ofimática en una única plataforma con integración en los portales corporativos.
- Estandarización de operativas.
- Evita dobles entradas y las replicaciones, si las hay, son responsabilidad del ERP.
- Seguimiento de las evoluciones tecnológicas.
- Adaptación a las normativas legales

Obviamente también tiene inconvenientes. Consulte el apartado dedicado a la eterna polémica entre programación a medida o ERP.

7. Los ERP como elemento de reingenieria.

La implementación de un ERP es una forma fiable y cara de hacer reingeniería. Es un método radical: sustitución de los sistemas heredados por un nuevo programa que aporta la mayoría de los procesos que gestión informática que la empresa necesita.

La instalación de un ERP puede ser un revulsivo necesario en una compañía dormida.

La implementación del ERP puede ser monofásica o por fases. La implementación monofásica, impensable hace pocos años es hoy día posible y alcanzable.

¿Qué es mejor? Como siempre, hay que valorarlo. Hay ventajas e inconvenientes:

- La implementación monofásica ahorra muchos interfaces con los sistemas que, en el caso de ser por fases, deben ir sustituyéndose paulatinamente.
- Si escoge implementar en una fase, debe utilizar más tiempo de preparación y pruebas minuciosas antes de lanzarse a dar el paso en real.
- El arranque en una fase exige un plan de contingencias a prueba de bombas y una cultura de trabajo en grupo no siempre disponible en las empresas.

Y, recuerde, hay verdades universales, arranque en una fase o en varias:

- Necesidad de un apoyo decidido por parte de la Dirección.
- Amplio conocimiento de la cultura corporativa.
- Motivación de los usuarios implicados.
- Implicación de los antiguos informáticos. Conocen la empresa y sus procesos informáticos como nadie y se merecen vivir el futuro y no ser penalizados atándoles al cuello la piedra de los sistemas que van a ser sustituidos.
- Necesidad de una consultoría externa. Aportará metodología, recursos ajustables y conocimientos. Pagando, claro.
- Necesidad de que el Know-how quede en la compañía y no en la consultoría externa.
- Necesidad de formar a fondo y concienzudamente a los usuarios del nuevo ERP.

8. Los ERP's como forma de adaptarse a normativas legales y a la evolución tecnológica.

Una de los grandes argumentos a favor de los ERP's es su adaptabilidad a la evolución tecnológica y a los cambios de normativas legales.

En este último ámbito, los ejemplos son tantos como los cambios producidos. El cambio del ITE al IVA en su momento, la superación del efecto 2000 y la adaptación al EURO son ejemplos del pasado.

Otros ejemplos pueden ser la adaptación a la regulación 2002/3622 del mercado común sobre la aplicación de normas contables (IAS – Internacional Accounting Standards), pensadas para poder comparar balances entre países adoptando unos estándares comunes y el nuevo Acuerdo Normativo de Capital denominado Basilea II para la gestión de capitales y aplicable desde el 2005 a las empresas de la UE que cotizan en bolsa.

Y en el campo tecnológico, como los ERP's tienen muchos clientes, evolucionarán seguro con las nuevas tecnologías. Seguramente no de forma tan rápida como Vd., querría (¿lo necesita?) pero de forma segura y cuando la tecnología este madura.

9. ¿Programación a medida o ERP?

El mundo idílico de los ERP's tiene, como ya hemos visto, contraprestaciones importantes:

- La inversión inicial es alta.
- Habrá de pagar una fuerte cuota de mantenimiento anual.
- Necesita de consultaría externa casi de forma permanente con el peligro de las decisiones las tome la consultora y no el personal interno.
- Son sistemas "pesados" en recursos informáticos.
- La organización debe adaptarse, en parte, al ERP para minimizar coste de personificación y desviarse lo menos posible del estándar. Esto es, por si solo, una ventaja y un inconveniente.
- No elimina la dependencia de las personas: el know-how informático-organizativo de la compañía lo tiene o el personal interno o la consultaría.

Ojo con este último dilema. Una de las teóricas ventajas de una solución ERP es que permite reducir la plantilla de informáticos internos. Pero es siempre a costa de pagar consultoría. Y si al final los consultores externos son los únicos que conocen el funcionamiento de los sistemas de información de la compañía,... ¡está perdido! Le darán siempre soluciones, pero a un coste y con una dependencia externa que no sabrá nunca si las soluciones que le proponen es la que más le conviene a Vd. o a ellos.

Si coge una calculadora y hace la suma de los servicios de consultaría externa y de la cuota de mantenimiento del ERP, y la divide por un coste ponderado de analistas y programadores, se quedará maravillado con la cantidad de informáticos que podrá pagar. Y, paralelamente, por la cantidad de software a medida que podrá fabricar.

Como siempre habrá que analizar cada caso para saber que es lo que más conviene.

Una solución muy utilizada es usar el cuerpo del ERP parametrizando al máximo y programando el ERP a medida lo mínimo posible y utilizar aplicaciones periféricas para particularizar la solución, asumiendo el ERP la función de armazón de la arquitectura distribuida.

Y por favor, haga que su organización siga mi consejo: **tenga personal interno para dirigir técnica y funcionalmente el proyecto y no se entregue en manos de una consultoría externa**, a la que deberá contratar, de eso no se escapa, pero a la que no deberá permitir dirigir ni conocer exclusivamente la arquitectura informática de la compañía.

10. ASP's.

Los ASP's, (Application Services Provider) son empresas proveedoras de servicios informáticos.

Pueden llevar el Outsourcing hasta el límite: ellos son el departamento informático de la compañía y su centro de datos. Y es una de los caminos por los que una compañía puede llegar a un ERP.

Alquilar software a un ASP puede ser una buena solución para compañías que buscan ahorran tiempo y dinero (?) a la hora de implementar aplicaciones.

Los ASP's se han visto potenciados por la llegada de Internet ya que esta plataforma a permitido un acceso más rápido, general y económico al software que alquilan.

La oferta de ASP's es muy amplia y difícil de catalogar. En el fondo, no es más que una redefinición de Outsourcing.

Hay proveedores ASP de:

- Aplicaciones.
- Servicios de Infraestructura, comunicaciones en particular.
- Housing.
- Software básico, etc..

En cualquier caso, es claro que este concepto es no es de diseño y que, por tanto, podemos no volver a referirnos a él en el resto del nuestro viaje.

Dos Gotas de Reingeniería de Sistemas

1. Introducción.

La reingeniería de sistemas, aprovechando las aplicaciones distribuidas por Cliente/Servidor basado en Sistema operativo y/o en Internet, permite rediseñar Sistemas de Información ampliando sus posibilidades y adaptabilidad.

El tema lo trataré en profundidad al final del libro aprovechando las experiencias del viaje a través del diseño.

Sin embargo, creo que no se puede acabar un ciclo de introducción a los sistemas distribuidos sin hablar dos palabras sobre este tema.

2. Evolución por escenarios.

La situación de un sistema informático y sus posibles caminos de evolución se ha representado tradicionalmente por el **Esquema de Escenarios** de la figura.

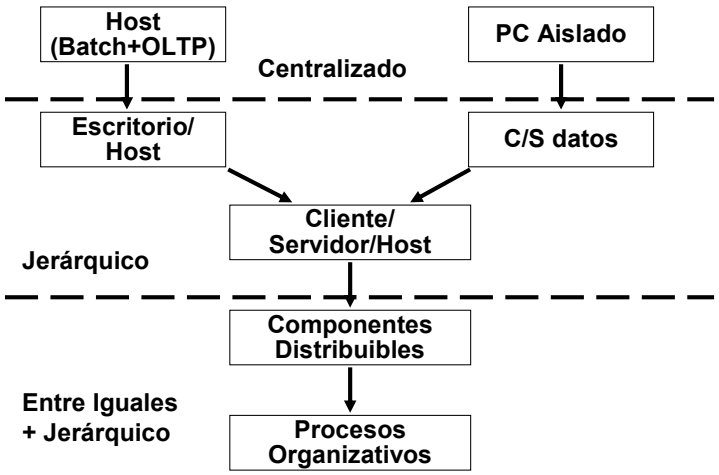


Figura 98. Evolución por Escenarios

El **Escenario Centralizado** supone la utilización de un HOST como único soporte de las aplicaciones de la compañía. Si existen PC's, solo están como elementos aislados en funciones especificas. Personalmente creo que es un escenario muy poco frecuente actualmente.

El siguiente escenario aparece con la interconexión del HOST y los PC's a través de redes y comunicaciones.

En este escenario, las aplicaciones son jerárquicas por lo que este escenario se denomina genéricamente **Escenario Jerárquico**.

Se presentan tres estadios:

- **Escritorio/HOST,** donde las aplicaciones continúan siendo Jerárquicas, pero hay datos que se incluyen dentro de los paquetes de ofimática (de ahí lo de escritorio) para su gestión especifica por los usuarios).

- **Cliente/Servidor de datos.** Cuando los PC's se integran en red, lo primero que se hace es compartir impresoras y datos utilizando los servidores de estos tipos incluidos en todas las redes.
- **Cliente/Servidor/HOST.** Las aplicaciones departamentales y de cliente se integran utilizando técnicas Cliente/Servidor, bajo sistema operativo o Internet, con el HOST. Las aplicaciones resultantes son inicialmente jerárquicas, de ahí el nombre general de este escenario.

El siguiente paso, no siempre recorrido, es realizar aplicaciones entre iguales, es decir, del mismo nivel que las centralizadas. Estas aplicaciones conviven con las jerárquicas anteriores. Solo en casos muy extremos y justificados las antiguas aplicaciones centralizadas con sustituidas totalmente y el HOST desaparece.

Este escenario se denomina **Escenario entre Iguales y Jerárquico.** Presenta también dos estadios:

- **Escenario de Componentes Distribuibles**, en el cual los componentes de las aplicaciones pueden repartirse por la plataforma con criterio distribuible (recuerde que es una situación menos fuerte que distribuida). La única limitación es que los elementos distribuibles estén situados en localizaciones en los que funcionan.
- **Procesos Organizativos**. En los cuales los procesos de negocio están distribuidos por toda la plataforma. Son las técnicas de Groupware como integradoras de procesos de negocio que engloban a más de un Dpto. y plataforma.

Este marco de escenarios tiene una doble utilidad:

- Situar la plataforma de una compañía en un marco comparativo claro.
- Plantear un marco de evolución de la plataforma. Aquí conviene aclarar que no necesariamente la mejor plataforma para una compañía es el último estadio. La mejor plataforma es aquella que la compañía necesita y que le sale más rentable en el conocido equilibrio prestaciones/coste.

La llegada y la popularización de Internet la supuesto una evolución en el planteamiento de escenarios que no lo hace tan lineal en su evolución. El estadio final constituido por los procesos organizativos ya no es la única posibilidad; y el camino puede recorrerse por más de una vía.

Es un esquema de escenarios no tan consensuado como el clásico pero que según mi opinión tiene el aspecto de la figura.

Como observarán han aparecido nuevos estadios en los estados generales del escenario.

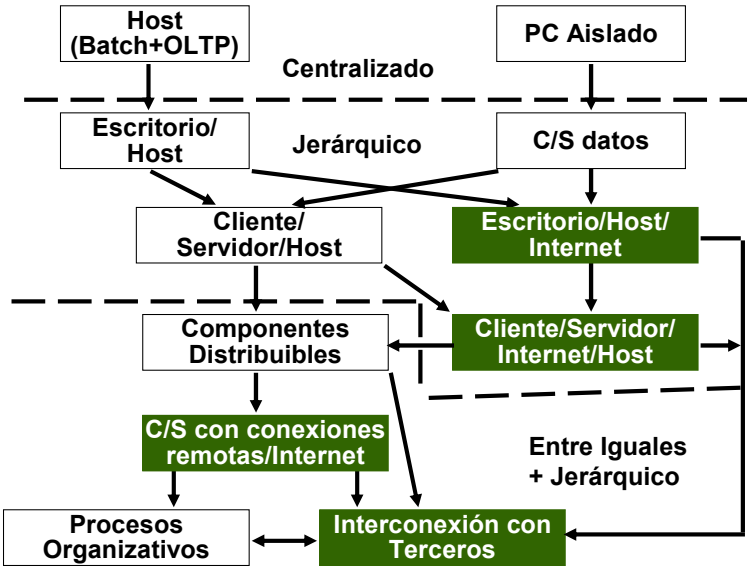


Figura 99. Esquema de escenarios evolucionado

En el Escenario Jerárquico han aparecido dos nuevos estadios:

- **HOST/Internet**. En el cual las aplicaciones centralizadas dejan su información disponible a través de una WEB o la emulación de terminal se traslada a la WEB. Ha sido un escenario muy interesante para empresas que no habían entrado en C/S por diversas razones.
- **Cliente/Servidor/Internet/HOST**. La distribución Cliente/Servidor se la ampliado con aplicaciones Internet.

Y en el Escenario entre Iguales + Jerárquico han aparecido otros dos estadios:

- **Cliente/Servidor con conexiones remotas e Internet**. En el cual, además de haber aplicaciones Internet clásicas, existen aplicaciones Internet activas y aplicaciones Cliente/Servidor que utilizan Internet como transportista.
- **Interconexión con terceros** a través de Internet. Este estadio, de hecho, existe también en el Escenario Jerárquico; no he incluido otras posibles evoluciones que acaben en este estadio para no complicar más la figura.

3. El flujo de los sistemas de información hacia la integración.

El flujo y la evolución de los Sistemas de Información hacia el Sistema Integrado pueden esquematizarse en el gráfico de la figura.

El Sistema Integrado se representa en el centro como el objetivo a conseguir.

Los posibles modelos de sistemas de información orbitan a su alrededor. Las flechas de un solo sentido indican vías de evolución. Las flechas de dos sentidos indican funcionamiento integrado.

Empecemos por analizar el mundo del HOST. El HOST puede evolucionar de tres formas:

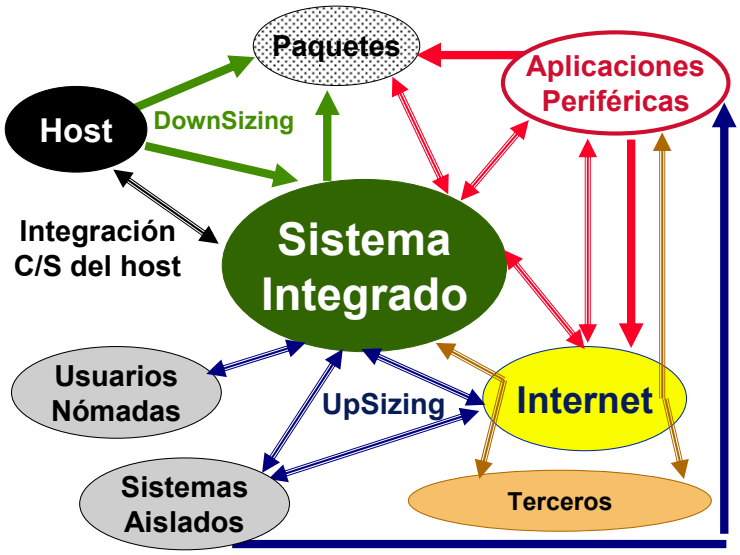


Figura 100. Flujo de los Sistemas de Información hacia la integración

- Integrándose en Cliente/Servidor con el resto de plataformas.
- Sustituyéndose por Downsizing. Hay dos posibilidades.
  - Sustitución de las aplicaciones corporativas por aplicaciones distribuidas. Esta opción solo ha tenido lugar en contadas ocasiones y en entornos donde por su naturaleza existe una motivación clara. Siempre se ha puesto como ejemplo típico la gestión de una compañía de paquetería de ámbito multinacional.
  - Sustituyendo las aplicaciones corporativas de siempre por un paquete de alto nivel tipo ERP. Este camino supone para la compañía que lo inicia una apuesta más allá de la informática: se está apostando por un modelo de organización empresarial en el cual el paquete informático no es más que la herramienta. Fundamental, eso sí.

El este último caso, de hecho, el sistema integrado es el paquete.

Las aplicaciones periféricas desarrolladas como apoyo y complemento de las corporativas y como soporte de las departamentales, pueden integrarse básicamente de tres formas:

- Utilizando un paquete que se integre como una pieza más del sistema.
- Realizando aplicaciones distribuidas Cliente/Servidor, integradas opcionalmente a través de Internet. Este caso, en que Internet es una vía de transporte, se representa en el esquema por las dos flechas que se unen dentro del ámbito de Internet.
- Realizando aplicaciones Internet, pasivas o activas.

La interconexión con terceros se realizará en la inmensa mayoría de casos a través de Internet con gran presencia de Servicios WEB o intercambio desacoplado de interfases en formato de ficheros.

Por último los sistemas aislados se integran dentro del sistema mediante una etapa de Upsizing.

Recuerde que cuando un sistema aislado se integra en el global es conveniente instalarle la plataforma estándar, incluyendo la plataforma ofimática de la compañía.

Esta operación puede no ser trivial y normalmente pide acciones de los administradores de sistema y cambios en la información particular del usuario, La carga de trabajo y los cambios de hábitos del usuario deben ser evaluados con precisión ya que es caso contrario el riesgo de problemas en coste, tiempo y roces con el usuario es muy alto.

4. Los ERP como elemento de reingenieria.

Recuerde, por favor, este apartado del capítulo dedicado a los ERP's.

5. Un comentario final.

Hablar de escenarios y flujos es, a mi juicio, más formación de cultura general que necesidad. Pueden diseñarse sistemas distribuidos perfectamente y hacer reingeniería sin saber nada de escenarios y flujos.

Tópicos.

1. Introducción.

Para acabar esta primera parte repasemos los tópicos clásicos que siempre se han manejado dentro del mundo de las aplicaciones distribuidas.

Simplemente voy a relacionarlos ya que en la mayoría de los casos ya se han citado con anterioridad.

2. Ventajas y Desventajas de los Sistemas Distribuidos.

2. 1. Ventajas.

- Situar y obtener funcionalidad donde y cuando se necesite.
- Alta relación rendimiento / coste.
- Adaptabilidad a;
  - La evolución de los procesos de negocio.
  - El crecimiento.
  - La dispersión geográfica.
  - Estandarización.
- Disponibilidad. Es más fácil tener alternativas a las caídas de la plataforma (con todos los matices que hemos comentado con anterioridad).
- Democratización de la informática. Desde ordenadores pequeños y baratos se pueden acceder a gran cantidad de servicios.
- Gran disponibilidad de software construido.
- Satisfacción alta de los usuarios.
- Estandarización de las operativas debido a la implementación masiva de Windows e Internet.

2. 2. Desventajas.

- Administración de sistema compleja.
- Necesidad de diseño de consistencia.
- Diseño más complicado en las aplicaciones avanzadas.
- Necesidad de formación más alta en los informáticos (¿es esto una desventaja?).
- Contaminación por virus.
- Herramientas más complejas.

3. Costes.

Tradicionalmente se ha dicho que los sistemas distribuidos son más baratos que los centralizados. No estoy de acuerdo. Aunque de hecho, si lo estoy. Déjeme que le justifique esta aparente contradicción

El problema de evaluar costes en un entorno distribuido es muy complejo. El problema básico está en que la ponderación de los factores positivos y negativos depende de cada organización.

En una primera aproximación:

- Los costes de hardware son más bajos.
- Los costes de software son:
  - Iguales en las aplicaciones RAD.
  - Mayores en las aplicaciones avanzadas.
- Los costes de administración son mayores.
- Los costes de formación son mayores.
- Las ventajas organizativas ponderan a la baja los costes.
- La fiabilidad de los elementos de sistema y de las herramientas de desarrollo es menor en los sistemas distribuidos.

Cuide con gran atención los dos costes escondidos:

- La formación de personal.
- La administración de sistema.

Sin embargo:

- La estandarización hace bajar los costes de desarrollo y administración.
- La alta disponibilidad de componentes fabricados baja considerablemente los costes de desarrollo.
- La oferta de profesionales en mucho mayor, por lo cual el mecanismo de oferta y demanda de la economía de mercado fuerza a la baja los costes de mano de obra (por desgracia para Vd. y para mi).

Todo ello puede ser esquematizado en una balanza de costes como la de la figura.

Como ve, a priori, mi opinión contraria a la corriente de opinión a favor de que los sistemas distribuidos son más baratos está plenamente justificada.

Pero con formación y experiencia Vd. podrá sacar tal provecho a los componentes fabricados y a la estandarización que bajará los costes reales de forma tal que mí contra opinión de que son más baratos se justificará también.

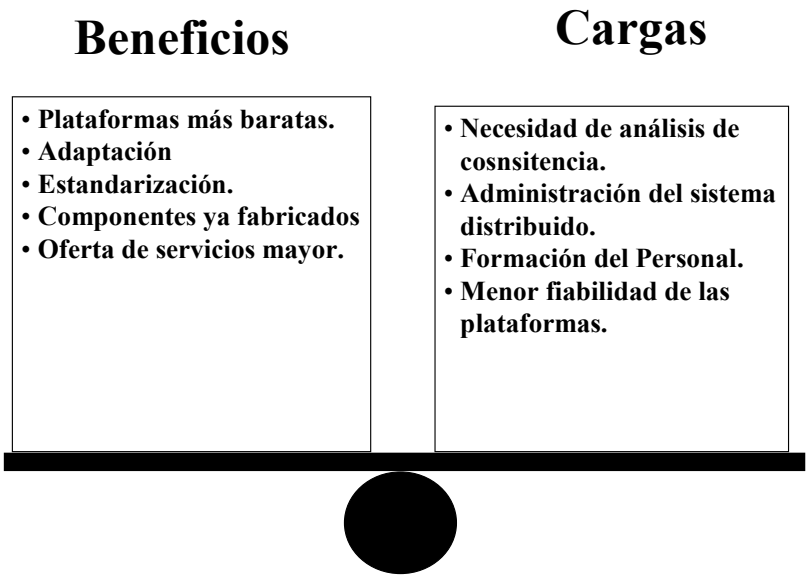


Figura 101. La balanza de costes.

De hecho, la evolución de costes presenta el aspecto de la figura donde los costes iniciales más altos en los sistemas distribuidos se flexionan a la baja de forma notable aplicando formación y reusabilidad. A estos factores añada el otro componente crucial: las ventajas que su organización puede sacar a la

adaptabilidad y resto de ventajas organizativas de las aplicaciones distribuidas que, ¿como se miden genéricamente en dinero?

La pregunta clave es: ¿como se calcula el punto X de inflexión del coste? Lo siento, no lo conozco ninguna fórmula. Si Vd. La consigue, publique un libro, ¡será un Best-Seler!

A nivel de proyecto, si sé la respuesta: un estudio de costes bien hecho.

Me deja, finalmente, que le diga un resultado empírico que no se justificar más que por mí (remarco lo de mí) propia experiencia: normalmente se ve clarísimo lo que es distribuido y no que no lo es.

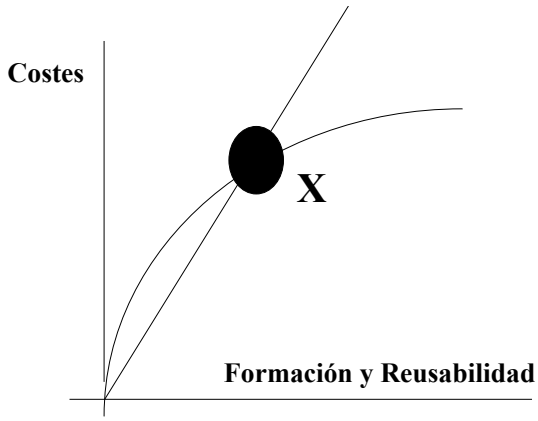


Figura 102. Comparación de costes.

INDICE DE LA PRIMERA PARTE

INDICE DE FIGURAS DE LA PRIMERA PARTE