Qué es una aplicación?

De una forma básica, podríamos definir una aplicación o programa informático como *un conjunto de instrucciones que ejecutan una tarea dentro de un ordenador.*

Los programas informáticos nacieron como procesos implementados en las primeras máquinas de los años 70, pero pronto pasaron a ser distribuidos como *software* independiente en formato físico (diskettes, cd's). Al adquirir una copia del programa deseado, se introducía físicamente en el ordenador y se ejecutaba (normalmente con instalación previa).

Éstos programas que requieren la copia total o parcial en la máquina del usuario son conocidos como *aplicaciones de escritorio*, y son los que hemos estado utilizando hasta prácticamente hoy en día.

Funciona el programa en mi ordenador?

Los programas informáticos han evolucionado enormemente desde sus inicios: hemos pasado de interfaces de líneas de comando a Interfaces Gráficas de Usuario (GUI, en inglés), de lenguajes de programación de bajo nivel (difíciles) a lenguajes de alto nivel… pero aún tenemos que cargar con algunos de sus problemas, entre ellos el de la compatibilidad de software.

¿Cuántas veces hemos echado en falta un programa en una arquitectura o sistema operativo? No hay 3DStudio para MacOSX, no hay Photoshop para Linux, no hay iLife para Windows… y un largo etcétera. Son problemas que se ven todos los días, y ante los cuales sólo hay dos soluciones:

- Una, por parte de los usuarios, es optar por alternativas: Gimp en lugar de Photoshop, Maya en lugar de 3DStudio, OpenOffice en lugar de Microsoft Office, etc…
- Otra, por parte de los programadores, sería la de ponerse de acuerdo y realizar aplicaciones multiplataforma.

La primera, aunque es viable, a veces supone tener que renunciar a características y utilidades de los programas ausentes que en ocasiones no se pueden permitir. Es la causa, o una de las causas, por la que muchas empresas no usan los programas de gestión disponibles para MacOSX, por ejemplo.

La segunda también es viable, pero mucho más difícil de llevar a cabo. Debería de existir una especie de "estándar" o conjunto de normas que todas las aplicaciones siguieran para asegurar la mayor compatibilidad posible.

Por lo tanto, la respuesta a la pregunta *¿Funciona el programa en mi ordenador?* sería: no, es imposible asegurar que un programa funcione en cualquier ordenador.

Puedo ver esta página web en mi ordenador?

El nacimiento de la Web posibilitó el acceso masivo a la llamada Tecnología de la Información. El uso de un lenguaje de texto y vínculos (enlaces) fue un acierto que hizo que todos los navegadores, todos los sistemas operativos, todas las arquitecturas; en definitiva, todo el mundo, pudiera participar de una red interconectada de forma transparente al usuario.

Sin embargo, en la Web no tardaron en aparecer algunos de los mismos problemas que aparecieron en el software de escritorio, entre ellos, el de la terrible incompatibilidad.

Hubo una oscura época en la cual muchas páginas sólo eran compatibles con el nefasto Internet Explorer (gracias, Microsoft), y cuando nos empezamos a librar de esa losa, apareció Flash Player para tomar el relevo (gracias, Adobe).

Afortunadamente, y gracias a que las cosas se hicieron bien desde el principio, en la Web existen unos estándares, unas normas, reguladas por un organismo internacional (W3C) que vela porque casos como el de IExplorer y Flash se vayan dejando atrás.

Son ellos los que ahora mismo están moviendo el tema de HTML5, con el que conseguirán eliminar las barreras que ahora mismo hacen que ver un vídeo en Flash sea una tortura fuera de Windows, o que los formularios de algunas webs de diputaciones provinciales sólo funcionen con Internet Explorer.

Con lo que si nos preguntan *¿Puedo ver esta página web en mi ordenador?* podríamos responder que sí, existen mecanismos que pueden asegurar que una página web puede verse sin problemas en cualquier ordenador.

Un término medio: RIAs

Existen algunas webs que necesitan algo más que mostrar los contenidos de forma estática, a las cuales nos referimos como webs *dinámicas*.

En algunos casos, el dinamismo requerido no es ciencia espacial (actualizar un contenido según el día de la semana, avisar al usuario si no ha rellenado un campo correctamente...), pero en ocasiones las acciones que se realizan después del click en el botón de "enviar" son de una complejidad comparables a las de las aplicaciones de escritorio (nos referimos a intranets, áreas personalizadas de un portal web, tiendas online, etc).

Es entonces cuando estamos hablando de *Aplicaciones Ricas de Internet* (Rich Internet Applications, RIAs) o *Aplicaciones Web.* Una aplicación web va mucho más allá del enfoque de una página web estándar, llega al nivel de las aplicaciones de escritorio e incluso a veces las superan.

En términos de arquitectura de software, las aplicaciones web suelen dividirse en dos bloques o capas:

- La capa de *Vista* o *Interfaz*, que es la que el usuario ve en su navegador y mediante la cual interactúa: es donde se encuentran las ventanas, botones, menús, imágenes...

- La capa de *Lógica de Negocio* (o *Servicios de Negocio*), es la encargada de ejecutar la acción que el usuario solicita (pulsando un botón, seleccionando una opción del menú) y de informar del resultado de la misma.

La capa de interfaz se maneja desde la máquina local del usuario, mediante su navegador web, mientras que la lógica de negocio está centralizada en un servidor. Una aplicación de escritorio se ve como un *paquete* que, mediante instalación o sin ella, se ejecuta de forma íntegra en el ordenador de cada uno.

Qué ventajas aportan las RIA? Y qué desventajas?

- *Compatibilidad*: al seguir los estándares de la W3C para el desarrollo de la capa de interfaz, se podrá ejecutar sin problemas en Firefox, Safari, Opera, Chrome… hasta en Internet Explorer :) con independencia del sistema operativo utilizado, claro. La capa de lógica sólo debe preocuparse de funcionar bien en la máquina servidora.

- *Escalabilidad*: gracias a la diferenciación Interfaz-Lógica, se puede cambiar el aspecto visual de la aplicación de forma independiente a las "tripas" que están en el servidor, e incluso se

podrían tener varias versiones distintas de la capa visual funcionando a la vez con la misma capa de lógica.

- *Economía* de recursos: al estar la lógica de negocio (la parte "dura" de la aplicación) centralizada en el servidor, no se necesita que la máquina del cliente tenga unas características técnicas elevadas. Un equipo de hace varios años podría manejar sin problemas una RIA actual.

De un plumazo, se solucionan muchos de los problemas que tenemos al usar aplicaciones de escritorio.

Pero como no podía ser de otra manera, no es oro todo lo que reluce, y el uso de RIA's también implican ciertas desventajas frente a las aplicaciones de escritorio:

- *Tipología*: no todos los tipos de aplicaciones de escritorio se pueden implementar usando RIA's. Hablo de programas de diseño, fotografía, videojuegos, etc. Para poder hacer un 3Dstudio online se necesitaría una conexión de banda *muy* ancha y un servidor con una potencia *bastante* alta.

- *Dependencia de la red*: absoluta y completa. Si un día que necesitas ejecutar la aplicación se ha cortado la línea no podrás siquiera acceder a la pantalla de bienvenida.

- *Contenidos Multimedia*: pese a que últimamente se están mejorando a pasos agigantados, los anchos de banda y los materiales utilizados en los cableados hacen que acceder a contenidos de audio o vídeo pueda ser un suplicio, por mucho streaming y técnicas de compresión que se usen.

Tienen futuro las RIAs?

Las aplicaciones web existen desde hace tiempo. El sector del software de gestión se está moviendo mucho hacia ése lado, y es de esperar que según se mejoren las condiciones de conexión y la cultura empresarial vaya cambiando, las posibilidades aumenten aún más.

Sin embargo, bajo mi punto de vista, hay dos acontecimientos en los que las RIA's pueden encontrar un nuevo impulso que podría consolidarlas como la gran alternativa al software de escritorio:

- Por un lado está el lanzamiento de ChromeOS, el sistema operativo distribuido de Google. La filosofía de "todo está en la nube" favorece que las aplicaciones que se ejecuten sean en su inmensa mayoría aplicaciones web.

- Y por otro, la llegada de los dispositivos móviles: la compatibilidad que se gana en los sistemas operativos de escritorio también se gana aquí. A día de hoy, tanto el iOS de Apple como el Android de Google, el WebOS de Palm y el Windows Phone de Microsoft, aceptan o van a aceptar las aplicaciones programadas con estándares HTML de W3C, que se comunicarían con un servidor de forma transparente al usuario, el sistema y el dispositivo.

Conclusión

En definitiva, tenemos una alternativa al desarrollo de software de escritorio usando las ventajas que ofrece la tecnología web, y que solucionaría varios de los problemas críticos que hemos venido arrastrando desde hace años.

Con el lanzamiento de ChromeOS y la necesidad de unificar las tecnologías de desarrollo de aplicaciones para los dispositivos móviles, las RIA's pueden encontrar un impulso del que hasta ahora han carecido, convirtiéndose en una alternativa sólida para el desarrollo de aplicaciones complejas.

1 - partes de una aplicacion web
    1.1 - capa de vista
    1.2 - capa de logica
    1.3 - servidor de aplicaciones
    1.4 - bd

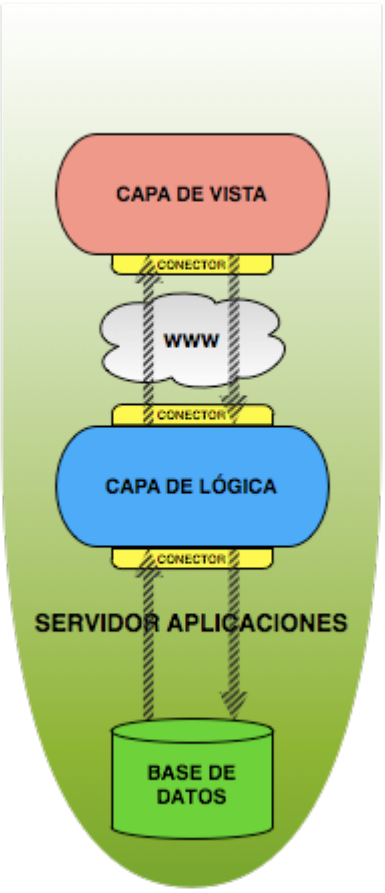
2 - herramientas de desarrollo de aplicaciones web

3 - ejemplos de aplicaciones web

1. Partes de una aplicación web

Como ya hemos apuntado, las aplicaciones web pueden verse como una suma de la Capa de Vista + Capa de Lógica. Pero para que una aplicación web funcione necesita más cosas, necesita *algo* que la mantenga en funcionamiento y otro *algo* que sirva para almacenar, recuperar, actualizar la información que el usuario le está pidiendo constantemente.

Éstos dos *algos* de los que hablamos son el *Servidor de aplicaciones* y la *Base de datos,* que forman junto a las dos capas de Vista y Lógica el esqueleto típico de una aplicación web:



Siguiendo el esquema de la figura, vemos cómo las capas de Vista, Lógica y Base de Datos están conectadas y se van pasando información, peticiones, etc, todo ello apoyándose en la ejecución del Servidor de Aplicaciones.

La comunicación entre la capa de Vista y Logica se realiza a través de la Web, mientras que entre la Lógica y la Base de Datos puede realizarse bien por Web o bien teniendo instaladas ambas en la misma máquina servidora.

En la figura, el Servidor de Aplicaciones envuelve el resto de elementos, ya que es él quien pone en marcha la aplicación, haciendo que todas las capas entren en funcionamiento (estén *encendidas* o *arrancadas*, por decirlo de alguna forma).

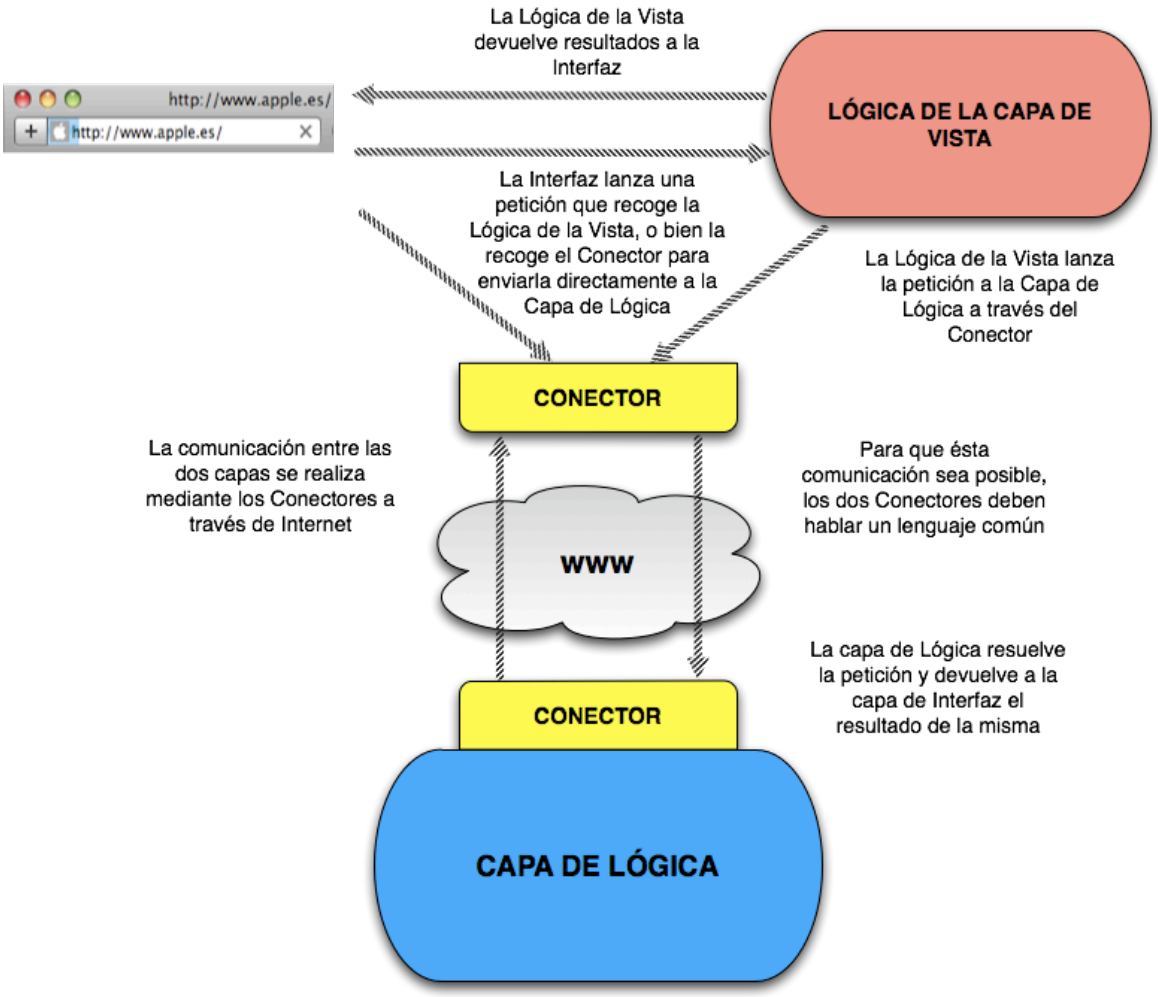
Finalmente, la comunicación entre las capas debe realizarse mediante unos *conectores* que dependen de la arquitectura con la que está construida la RIA.

A. Capa de vista

La capa de *Vista* o *Interfaz* es la encargada de proporcionar al usuario los controles y la información necesarios para interactuar con la aplicación web.

Ésta interfaz se carga en el ordenador del usuario, por lo que una de las cosas a tener en cuenta durante su desarrollo es que sea lo menos pesada posible, para que la velocidad de ejecución no se vea afectada.

Entrando en el detalle del esquema anterior, podríamos abstraer la capa de Vista de la siguiente forma :



Éste podría ser el esquema básico de funcionamiento de cualquier RIA:

- De una parte tenemos la Interfaz que ve el usuario en su navegador Web: es lo que aparece si abrimos Safari (o Chrome, o Firefox...) e introducimos la dirección de la página de entrada a la aplicación.

- El usuario tiene disponibles en ésa página todos los controles necesarios para usar la aplicación: menús desplegables, entradas de textos, selectores de imágenes, etc. La interacción con estos controles, bien sea seleccionando opciones de un menú o introduciendo texto en un campo habilitado para ello, son operaciones que no precisan de la capa de Lógica, sólo tienen como objetivo seleccionar datos que posteriormente se procesarán.

- Y a la hora de procesar esos datos, puede hacerse de dos formas: haciendo una petición directamente a la capa de Lógica, o pasando esos datos por un proceso previo, en una nueva unidad que podemos llamar *Lógica de la capa de Vista*.

Éste paso intermedio es necesario muchas veces para preparar los datos o efectuar comprobaciones previas a la llamada a la capa de Lógica: por ejemplo, si introducimos una fecha con formato incorrecto (algo así como *54-06/abcd*) no tiene sentido mandar ése dato a la capa de Lógica, debe ser la propia capa de Vista quien analice ésa fecha e informe al usuario que el formato no es correcto.

- Una vez preparados los datos, se mandan a la unidad que denominamos *Conector* para que se monte una *estructura* que contiene por una parte esos datos y por otra información sobre la conexión que debe ser transparente al usuario. Tras ello, el *Conector* enviará ésa estructura a través de Internet hacia la Capa de Lógica, en donde

esperará otro *Conector* que transformará la estructura de la petición en los datos que el usuario ha introducido para que sean procesados.

- Tras ése proceso, la capa de Lógica vuelve a montar una estructura mediante el Conector para enviarlo de vuelta a la capa de Vista, donde será tratada para informar al usuario del resultado de su petición.

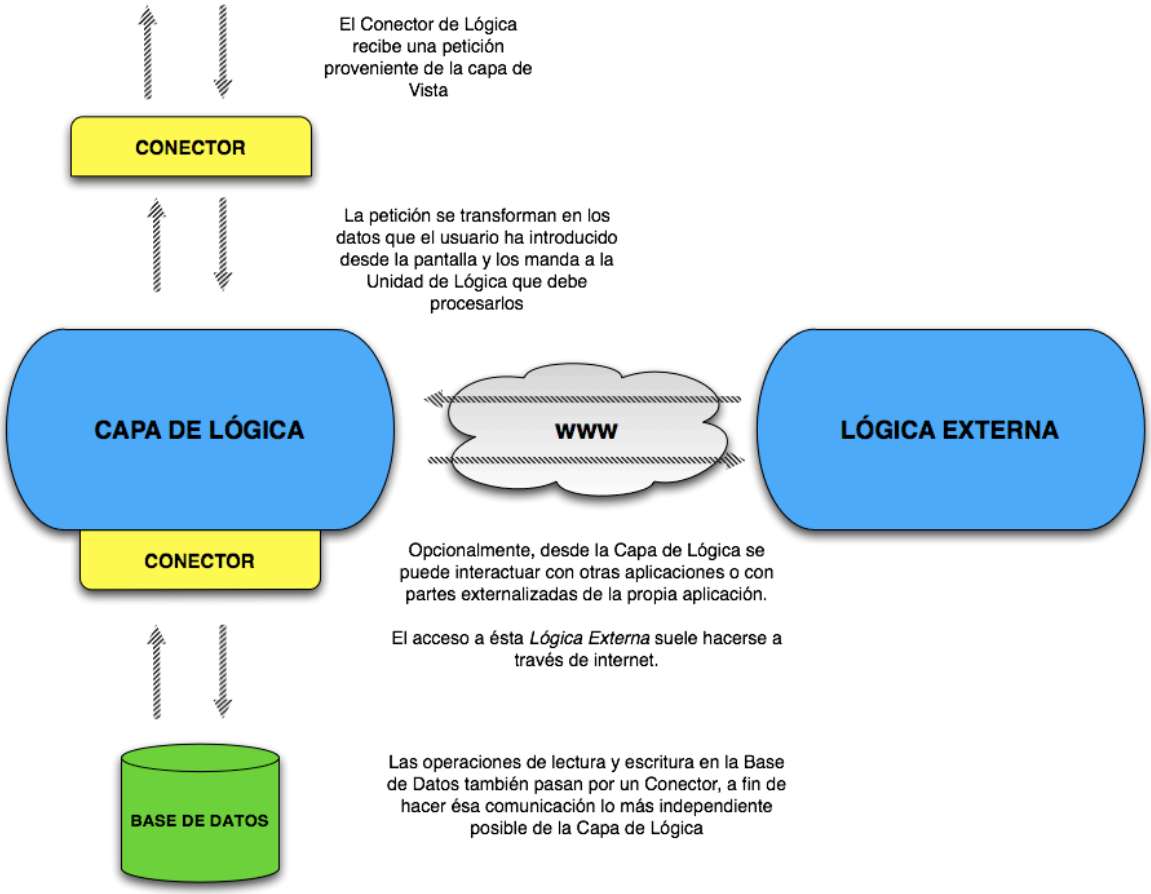
La manera de comunicarse con la capa de Lógica, de presentar los resultados en pantalla, de montar la estructura que los Conectores entiendan... todo ello debe estar formalizado y detallado, de modo que en toda la aplicación se haga de la misma forma. Este conjunto de reglas y de descripciones de la forma de trabajo en la capa de Vista (o en cualquier otra) es lo que se conoce como ***Framework***.

Un Framework (cuya traducción literal es “Marco de Trabajo”) describe cómo debe funcionar la aplicación y de qué forma se ejecutan las operaciones que el usuario no ve. El funcionamiento anteriormente descrito sería una visión general del Framework utilizado para una RIA.

Existen multitud de Frameworks o *modelos* de funcionamiento: *Cairngorm*, *MVC*, *Google Web Toolkit*, etc... La elección del Framework de una RIA debe hacerse desde el conocimiento de la tipología de la misma: por ejemplo, no tiene sentido aplicar Cairngorm (un Framework con una fuerte carga de lógica en la capa Visual) si la interfaz de nuestra aplicación son simples formularios de datos sin ningún proceso adicional, ya que recargaría en exceso la ejecución en el navegador web del usuario, dando una sensación falsa de lentitud (no es que la aplicación vaya lenta, es que hace muchas cosas que en principio no son necesarias).

El concepto de Framework no se restringe simplemente a la capa Visual. En la capa de Lógica también se formaliza la forma de hacer las cosas y de tratar las peticiones, con lo que podemos hablar de *Frameworks de Lógica*, en contraposición a los *Frameworks de Vista* que hemos nombrado anteriormente: *Spring*, *Struts*, *Zend Framework*, etc.

B. Capa de Lógica



La puerta de entrada a la parte de la aplicación web que tenemos alojada en la máquina servidora es el Conector de Lógica.

La misión del Conector es, al igual que el Conector de la capa de Vista, transformar la petición que nos llega en los datos que el usuario introdujo desde la Interfaz, y mandarlos a la *unidad* que debe procesarlos.

Ésta *unidad* tiene implementada la lógica de negocio, que es la propia de cada aplicación:

- Si nuestra aplicación es de gestión de una tienda de animales, la unidad de *Compra* debe encargarse de registrar el animal como vendido, efectuar el cobro, emitir una factura, etc…
- Si nuestra aplicación es de gestión de una tienda de azulejos, la unidad de *Compra* debe encargarse de realizar el pedido al proveedor, avisar al repartidor con una fecha y lugar de entrega, etc (además de emitir facturas y demás operaciones comunes).

Aquí se puede ver como dos conceptos comunes (la compra de *algo*) puede resolverse de formas diferentes dependiendo de la tipología de nuestra aplicación. La independencia de la capa de Lógica hace que sea posible crear capas de lógica de aplicaciones web con una misma estructura (he aquí nuestro Framework) y que cada una tenga su propia lógica de negocio.

En el lenguaje de las RIAs, es muy común hablar de *peticiones* a la hora de tratar la comunicación entre las distintas capas. Podemos entender una *petición* como una unidad de comunicación, una cápsula que contiene los datos propios de la petición más una información extra que se necesita para saber dónde hay que mandar esa petición, cuándo hacerlo, y qué se debe hacer con el resultado de la misma.

Por ejemplo, en una aplicación de gestión de una tienda de animales la operación de comprar una mascota podría generar una petición del tipo:

Nombre de la petición: Comprar_mascota

Datos de la mascota:
- **Raza:** Fox-Terrier
- **Edad:** 3 meses
- **Vacunado:** Sí
- **Precio:** 700 €

Datos del comprador:
- **Nombre:** Juan Pérez Arribas
- **DNI:** 53357651L
- **Domicilio:** Calle República 4
- **Teléfono:** 651987962

Unidad de Lógica: Unidad_de_compra_de_mascota
Si la compra va bien: emitir factura y avisar con un sms al nuevo dueño
Si ocurre algún error en el proceso: mostrar una ventana avisando del error y cancelar la compra

Desde la capa de la Vista se crearía ésta petición en el Conector deVista, y se enviaría a través de internet hasta el conector de Lógica, el cual separaría los datos que ha introducido el vendedor (los datos de la mascota y del comprador) de los datos que la aplicación necesita para saber qué hacer con la petición (el nombre, la Unidad de Lógica que va a ejecutar la compra, qué hacer si va bien o mal, etc…).

Uno de los factores que distingue a una RIA bien implementada es la de la coherencia en la creación de las peticiones. Si la forma de pedir a la aplicación que haga algo es siempre la misma, será más rápido y sencillo para ésta ejecutar las acciones, y también hará que realizar cambios sea más sencillo.

Los *Conectores* son una parte fundamental de la estructura de una RIA. Si imaginamos el tráfico de peticiones entre las capas de Vista y Lógica como una empresa de mensajería, el *Conector* sería el encargado de decidir qué tipo de albarán debe rellenarse por parte del repartidor y del cliente, qué transporte usar, cuáles son los límites de peso, etc…

Por norma, deben ser lo más generales posible. Es decir, no debemos tener un Conector para peticiones de compra, otro para resultados, otro para llamar a otra aplicación, otro para recibir datos o comunicarse con la impresora…

Lo ideal sería tener un Conector por cada canal de comunicación entre las capas (Vista – Lógica, Lógica – BD, Lógica – Lógica externa…) ya que de este modo fortalecemos la coherencia del software. Desde el punto de vista de la programación, el saber que cualquier interacción con las capas va a hacerse siempre en un mismo punto y de la misma forma hace que el crecimiento y mantenimiento de la aplicación sea muy fácil.

El Conector también es importante desde un punto de vista de arquitectura de software. Aunque puede programarse toda una RIA con la misma tecnología, es bastante normal que cada capa esté escrita en un lenguaje de programación distinto. Esto es así porque hay lenguajes más adecuados que otros para implementar la capa Visual o la capa de Lógica. En éstos casos los Conectores deben, además de gestionar la comunicación, transformar la propia petición de un lenguaje de programación a otro.

Una vez la petición llega a la unidad de Lógica correspondiente, se ejecuta la lógica de negocio que tenga implementada. Típicamente, ésta lógica consistirá en tratar los datos según el negocio e interactuar con la Base de Datos (guardando los datos, actualizándolos, etc).

La conexión con la Base de Datos también debe hacerse a través de un Conector. Aunque es posible ejecutar instrucciones de SQL (el lenguaje de la base de datos) directamente desde la Unidad de Lógica, el uso de un Conector posibilita alternar entre varios motores de Base de Datos, independizar la tecnología de la capa de Lógica, etc…

Adicionalmente, es posible que debido a la especificación de la lógica de negocio que atiende la Unidad de Lógica sea necesario comunicarse con otra aplicación web a través de internet. La forma de resolver ésta comunicación es verla como otra capa: definimos un Conector, creamos una petición y la enviamos para que la otra RIA la resuelva. Lógicamente, el Conector y la petición serán distintas de las del resto de nuestra aplicación web, ya que debe ser coherente con la aplicación con la que se comunica. En términos de arquitectura de software, se suele hablar de *Servicios Web*.

La visión por capas aquí vista es una aproximación global a toda la RIA, pero no es la única. Las aplicaciones web que se estructuran de ésta forma también suelen llamarse SOA (Service-Oriented-Architecture), aunque existen otras, como las arquitecturas orientadas a objetos (OOA).

Las ventajas de usar una SOA en el desarrollo de la RIA son los que ya hemos visto aquí:

- ***Tiempos reducidos en la realización de cambios***: la abstracción de los Conectores y Peticiones, así como de las distintas capas provocan que un cambio tenga poco impacto en la aplicación.

- ***Facilidad de evolucionar los modelos de negocio***: cuanto más abstraida esté la capa de Lógica, más fácil es añadirle cosas, cambiarle las que ya tiene...

- ***Facilidad para reemplazar elementos de las distinas capas sin alteración en el modelo de negocio***: mientras las comunicaciones se mantengan iguales en los conectores, la capa puede cambiar de forma o incluso cambiar enteramente sus partes.

- ***Integración de tecnologías distintas***: entre las distintas capas, volcándose las tareas de integración en los Conectores.

Con todo, podríamos resumir el flujo de una petición (esto es, lo que ocurre cada vez que el usuario interacciona con la aplicación) en los siguientes pasos:

1. El usuario selecciona un menú o pulsa un botón de envío de datos à desde la capa de Interfaz se recogen los datos que el usuario ha seleccionado y se mandan al Conector de Vista para que construya una petición. Los datos pueden haber pasado antes por la lógica interna de la Vista.

2. El Conector de Vista manda, a través de internet, la petición al Conector de Lógica. Una vez allí, se analiza la petición y se mandan los datos a la Unidad de Lógica correspondiente.

3. Dentro de la Unidad de Lógica, se tratan los datos conforme al proceso de negocio definido. En ése proceso es posible que se necesite interactuar con la Base de Datos mediante un Conector de Base de Datos; o con algún Servicio Web perteneciente a otra aplicación web, para lo cual también se usará otro Conector.

4. El resultado del proceso se devuelve al Conector de Lógica, que lo encapsulará en otra petición para devolver a la capa de Vista. Si se ha producido algún error durante el proceso se marcará en la petición.

5. El conector de Vista recibe, a través de internet, la petición de respuesta desde la capa de Lógica, en donde se comprueba la marca de error. Dependiendo de ésta, la aplicación se comportará de la forma en que se haya definido en la petición inicial (mostrando un mensaje de error si ha ido mal, o recargando la pantalla si ha ido bien, por ejemplo).

C. Servidor de aplicaciones

Otra de las partes fundamentales en la estructura de una RIA es el Servidor de Aplicaciones. La definición técnica de un servidor de aplicaciones sería la de *programa o software que proporciona servicios de aplicaciones a computadoras clientes que se conecten a él*.

De una forma más informal, podríamos decir que el Servidor de Aplicaciones es un software que permite que ciertas partes de una aplicación web sean accesibles desde internet.

Es decir, para que podamos acceder a nuestra RIA, debe haber un Servidor de Aplicaciones que la tenga *publicada*. Una vez está en marcha, podemos acceder a nuestra aplicación a través de internet. Normalmente, la parte que se publica es la de la capa de Lógica.

El término *Servidor de Aplicaciones* en principio no se restringe a una tecnología en particular, pero dado el éxito que ha tenido el lenguaje de programación Java, ahora mismo es sinónimo hablar de servidores de JEE (anteriormente J2EE). Pero, de igual modo, Microsoft también llama *Servidor de Aplicaciones* a su IIS (Internet Information Server).

El funcionamiento de los servidores de aplicaciones suele ser siempre el mismo: de un lado tenemos el Servidor, que actuará como un recipiente o contenedor de la aplicación. El proceso que introduce la aplicación en el servidor se conoce como *despliegue*. Una vez desplegada la aplicación (o aplicaciones), el servidor inicia las rutinas para la publicación en web, o *arranque.*

Existen multitud de servidores de aplicaciones: WebSphere (IBM), WebLogic (Oracle), JBoss, Zope, Base4…

La elección del servidor de aplicaciones que va a publicar nuestra aplicación depende de las necesidades de la misma: WebLogic o JBoss son servidores complejos propios de entornos de producción de grandes empresas. Para entornos más modestos, puede ser suficiente con Jetty o Tomcat (Apache) que, pese a no ser estrictamente servidores de aplicaciones, cumplen perfectamente con la tarea de publicar aplicaciones web.

D. Base de Datos

La finalidad última de las aplicaciones web suele ser el de gestionar la información almacenada en una base de datos.

Las bases de datos son un conjunto de datos almacenados en base a un esquema que permite operaciones de recuperación y almacenamiento de los mismos. Para ello, se ha definido un *lenguaje* que el motor de la base de datos entiende y sirve para interactuar con ella: SQL, lenguaje de consulta estructurado (structured-query-language).

Cuando una aplicación web accede a una base de datos, manda sentencias (frases) en lenguage SQL para realizar la operación de borrado, inserción o modificación que tenga definida según la lógica de negocio. Los datos que se utilizan se transforman en estructuras que tienen sentido dentro de la implementación de la capa de Lógica y se manejan en ella.

Aclaramos que estamos hablando de bases de datos *relacionales*, en las cuales los datos están enlazados entre sí mediante marcadores. Existen otros tipos de bases de datos, como las orientadas a objetos o declarativas, que no tienen cabida (o es meramente presencial) en el mundo de las RIAs.

Existen multitud de motores de bases de datos: Oracle, Informix, SQLServer, PostgreSQL, MySQL, SyBase… incluso podríamos citar aquí Filemaker o Access. Todos ellos comparten la sintaxis de SQL en su gran mayoría.

Pero ocurre que no todas las bases de datos (a partir de ahora obviaremos el término *motor de base de datos*) son apropiadas para cualquier aplicación web. Al igual que en el apartado de los Servidores de Aplicaciones, la elección del motor de base de datos a utilizar depende de las necesidades de la RIA.

Por ejemplo, MySQL y PostgreSQL son motores sencillos pero funcionales, que cumplen perfectamente para aplicaciones pequeñas. Sin embargo, el motor de Oracle es más robusto y tiene características que la hacen imprescindible en entornos de grandes empresas (por ejemplo la capacidad de hacer *flashbacks* : es posible recuperar la base de datos tal cual estaba hace unas horas, minutos, e incluso segundos).

La comunicación entre la RIA y la Base de Datos es parte importante de la estructura de la Aplicación Web. Pese a compartir la mayoría de sentencias de SQL, existen pequeñas diferencias entre los motores que obligan a dar un tratamiento específico para cada uno de ellos.

Siguiendo el modelo de capas que hemos visto hasta ahora, podemos considerar la comunicación con la Base de Datos como otra capa, definiendo un Conector que aísle a la capa de lógica de los detalles engorrosos de la llamada a la base de datos.

Éstos Conectores, en el ámbito de las bases de datos relacionales, son conocidos como ORM (object-relational-mapping, mapeo relación-objeto), y proporcionan a la aplicación una forma de acceder a los datos abstraída del motor… algo así como si el Conector dijera “pídeme los datos, que yo me ocuparé de hacerlo de modo que le guste a Oracle, Informix, o quien sea”.

Existen multitud de ORM para todas las plataformas: Hibernate, Ibatis, ObjectMapper, TopLink, SQLMapper… cada uno con sus características propias. Como siempre, la elección del ORM adecuado dependerá de las características de la RIA a la que dará soporte.