

Arquitectura de Sistemas: Un enfoque Evolutivo

(versión 1.2)

Puedes descargar la última versión de este documento de:

<http://www.morales-vazquez.com/fencasa.html>

José María Morales Vázquez

josemaria@morales-vazquez.com

Resumen. La mayor parte de las empresas modernas comienzan a subdividir las antiguas unidades de sistemas en dos grupos de trabajo bien diferenciados: aquellos que se encargan de la infraestructura y los que comienzan a denominarse arquitectos de sistemas. Este documento, una traducción al castellano casi literal de un white paper de Quidnunc, una empresa especializada en gestión de configuración, que explica la importancia de esta subdivisión y las pautas a seguir a la hora de diseñar la arquitectura de sistemas de una empresa.

1 Introducción

Entendemos por arquitectura en un proyecto informático a la disposición conjunta y ordenada de elementos software y hardware para cumplir una determinada función. No es difícil de comprender que si mezclamos arquitecturas distintas e inconsistentes sin ningún tipo de orden o planificación el proyecto se puede convertir fácilmente en ingobernable, tanto o más cuanto mayor sea la envergadura del mismo.

La mayoría de las organizaciones suelen favorecer (de forma planificada o no) unas configuraciones concretas. La arquitectura de cada empresa debería describir estas configuraciones y el entorno que facilite crear nuevas funcionalidades que encajen en ella, incluyendo directivas, componentes de software reutilizables, herramientas, etc.

Para facilitar que las nuevas funcionalidades de que dotemos a nuestra arquitectura sean consistentes con el sistema actual y las posibles modificaciones futuras de este, necesitamos conocer dicha arquitectura, pero es mucho más importante conocer la arquitectura operativa y organizacional de la empresa.

Una distinción importante es la que existe entre la arquitectura de una simple aplicación (micro arquitectura) y la que existe entre y a través de las distintas aplicaciones (macro arquitectura). No hace falta decir que esta última es la más compleja e importante.

1.1 Divide y vencerás

¿Cuál es el papel de la arquitectura en una organización? Imaginemos que nuestra organización consta de cuatro capas. La capa superior está formada por las actividades propias de la organización en sí misma. Debajo se encontrarían las aplicaciones informáticas que soportan y facilitan esas actividades. Por debajo de las aplicaciones se encuentra la arquitectura que facilita que estas se desarrollen y ejecuten. En último lugar, yace la infraestructura como el hardware o las redes físicas.

Esta subdivisión en cuatro capas nos facilita determinar el papel que desempeña la arquitectura dentro de una organización. Cada capa actúa como cliente de la capa inferior a ella y como servidor de la capa superior. Los arquitectos no deben de malgastar su tiempo en temas relacionados con la infraestructura, tales como el sistema operativo. La mejor forma de separar la arquitectura de la infraestructura es tener en mente el esquema de cuatro capas antes mencionado: la infraestructura debe de dar soporte a la arquitectura. Mezclar erróneamente conceptos de una y otra capa es un error muy común en muchas organizaciones.

Un error en el que no debe de caer un arquitecto de sistemas es ser demasiado preceptivo. Introducir demasiadas normas que creen una excesiva rigidez provocará problemas en el desarrollo de aplicaciones. Un buen arquitecto de sistemas debe de tener siempre en mente que su principal finalidad es permitir la creación de aplicaciones, facilitando la creatividad y la innovación de los creadores de las mismas.

La arquitectura de sistemas en los tiempos en los que sólo existían los mainframes era muy sencilla: existía un lugar para cada cosa y cada cosa tenía su lugar adecuado. Con el paso de los años y siempre en busca de una mayor flexibilidad se han ido in-

roduciendo estructuras cada vez mas y mas complejas: arquitectura cliente / servidor, arquitectura a tres capas, message brokers, data warehouses, objetos distribuidos, arquitecturas webs...

Un buen arquitecto debería de empezar por recordar que su trabajo es hacer la vida mas fácil a los desarrolladores, y no al revés...

Existe otra idea que subyace tras todos los enfoques: especialización: dividir los problemas en sus partes constituyentes y resolverlas separadamente con equipos de especialistas centrados en un área único. La especialización deja dos puntos sin respuesta: como dividir los sistemas para que puedan ser definidos separadamente y como unirlos posteriormente para formar un todo homogéneo. Estos son los principales retos de la moderna arquitectura de sistemas.

2 Arquitectura Evolutiva

Pero, ¿y desde el punto de vista de la empresa? ¿qué características debería de reunir la arquitectura de sistemas?

Si la mejora en los procesos y aplicaciones redundan en un mejor rendimiento de la empresa, y si para mejorar las aplicaciones necesitamos mejorar los sistemas, entonces la arquitectura de sistemas debería de ser el vehículo de desarrollo de ambos. En la práctica la arquitectura de los sistemas actuales constituyen, en muchos casos, grandes obstáculos para los dos.

A principios de los años 90, la arquitectura de sistemas no iba mas allá de una mera planificación. Definimos una arquitectura objetivo e ideamos una estrategia y una planificación para completarla dentro de unos determinados plazos de tiempo. La principal ventaja de este enfoque es que la hace comprensible a los ejecutivos, pues es similar a la forma en que tienen en dirigir sus negocios. El principal inconveniente es que no funciona. Comienza definiendo una arquitectura objetivo y esto es un error. El único objetivo que debe de tener en mente el arquitecto de sistemas es el de la organización para la que trabaja, si no tarde o temprano entrara en conflicto con el. La arquitectura del sistema debe de ser lo suficientemente flexible como para acomodarse a los cambios de objetivos de la organización. Esta es la clave principal para asegurar su longevidad.

La segunda clave a tener en cuenta es la que nos proporciona la mejor forma de medir la bondad de una arquitectura: la forma en que sustenta a las aplicaciones que sustentan a la organización. La mejor forma de verlo es estudiar, dada una nueva funcionalidad necesaria para nuestra empresa, como la arquitectura del sistema facilita su desarrollo e integración con el resto de las aplicaciones.

Los elementos claves que debe de cumplir nuestra arquitectura para facilitar el desarrollo de nuevas aplicaciones son: tener unas directivas claramente definidas pero no rígidas en exceso ni dictatoriales en cuanto al uso de determinadas tecnologías o fabricantes, favorecer el uso de aplicaciones que posean una funcionalidad base y sean personalizables por el usuario y facilitar el uso y desarrollo de componentes y plug-ins y aplicaciones que los admiten.

Este enfoque nos permite en la mayoría de los casos encontrar la forma mas rápida y sencilla de desarrollar una nueva funcionalidad para nuestro sistema en los casos en los que lo mas importante es tener una aplicación que nos haga lo que queremos y no tener la mejor aplicación que haga lo que queremos. Hoy en día, en la práctica, esta es la solución que necesitamos en la mayoría de los casos. Estos son los principios del enfoque que se conoce como evolutivo.

El modelo evolutivo, en el cual la arquitectura del sistema va adaptándose paso a paso, cada uno de ellos basado en los anteriores y siendo el mejor de todos los posibles que podemos dar en cada momento y no siguiendo un plan maestro tiene un poderoso antecedente: la evolución natural. Esta posee dos elementos cruciales: un método de producir variantes (la reproducción) y un método de elegir la mejor entre estas (la supervivencia de los mas fuertes).

Los métodos evolutivos son también muy populares en el desarrollo de software a medida: las aplicaciones suelen construirse mediante una serie de pasos consecutivos y no de una sola vez, reduciendo así considerablemente el riesgo de fallos y el coste de desarrollo. Estrictamente hablando, en este caso el termino evolución no está bien usado puesto que no existe ni variación ni selección. Un termino mas adecuado sería desarrollo incremental.

Al igual que en la evolución natural, las variantes en el mundo de la informática son abundantes y sólo los sistemas mas abiertos sobreviven. Es fundamental crear un entorno que propicie la tecno-diversidad en la arquitectura del sistema.

Una excepción a esta teoría la constituyen, en si mismos, los mainframes, los cuales han resistido mas de lo que se podía esperar a pesar, incluso, de la epidemia del año 2000. El problema es como crear un entorno que facilite la tecno-diversidad, donde las arquitecturas preestablecidas sobrevivan donde sea necesario pero sin bloquear el paso de los nuevos esquemas.

Las organizaciones con arquitecturas evolutivas poseen ciertos rasgos en común:

- Prefieren las directivas a los standards. Los standards reales son minimalistas y usualmente ‘de hecho’ como Windows. Las directivas pueden pasarse por alto si existe una razón lo suficientemente buena. La mayoría de las organizaciones mantienen demasiados standards motivados por la reducción de costes (por ejemplo, mantienen UNIX en el back-end para minimizar el coste de reeducación) pero fallan al ignorar el coste que supone forzar a determinadas aplicaciones que necesitan realmente tomar una línea diferente. La flexibilidad es una necesidad fundamental en la arquitectura de los sistemas modernos.
- Usan tecnología orientada a componentes. La historia de la informática describe un viaje inexorable hacia la especialización. Los componentes nos traen, por fin, la flexibilidad y posibilidad de reutilización del software que la orientación a objetos nos prometía desde hace tiempo
- Juzgan la arquitectura del sistema desde el punto de vista del usuario.
- Invierten en infraestructura. Ahorrar gastos en hardware o comunicaciones es, a menudo, un falso ahorro: un efecto de los días en que estas cosas eran caras. Ahorrar dinero ahora nos costará mucho mas dinero mas adelante.
- Reflejan la arquitectura de su organización en la arquitectura del sistema. Por ejemplo, organizaciones centralizadas necesitan un sistema centralizado mientras que organizaciones descentralizadas se adaptan mejor a sistemas distribuidos. Esto es una directiva mas que una regla.
- A la hora de elegir entre distintas aplicaciones toman como principales criterios la facilidad de uso y el impacto en el negocio.
- Eliminan los proyectos fallidos o débiles rápidamente. Cada peseta invertida en un proyecto débil o fallido es un dinero malgastado dos veces: aprende la lección, evita recriminaciones y corrige el problema.
- Valoran el capital intelectual. El principal activo de un departamento de arquitectura de sistemas son sus personas y los procedimientos que ellos conocen o han desarrollado. Es preciso cuidar apropiadamente esa aportación.

- Evitan innovaciones.

Estos puntos no pretenden ser una línea de actuación para beneficiarnos de una arquitectura evolutiva. Son una lista de observaciones procedente de organizaciones que manejan una arquitectura de estas características.

3 La Revolución de los Componentes

Dos décadas después de comenzar a andar el camino, por fin tenemos la tecnología suficiente para crear y ensamblar componentes que otras ramas de la ingeniería disfrutaban desde hace mucho tiempo.

Hasta hace sólo unos pocos años la industria del software atravesaba una grave crisis: las empresas demandaban desarrollos cada vez más y más complejos en plazos de tiempo cada vez más ajustados, y a precios más competitivos. La actividad de desarrollar software es ya de por sí lo suficientemente lenta y frustrante para tener que aguantar esto.

Repasando la historia nos encontramos con que otros sectores de la industria han atravesado antes por estos problemas. El ejemplo más claro lo tenemos en la industria automovilística. Hoy en día, el trabajo en las plantas donde se construyen los coches se limita al ensamblaje de componentes. Estos componentes son suministrados listos para su montaje por proveedores plenamente especializados en la fabricación de los mismos que posiblemente no tengan ni idea de los conocimientos necesarios para hacer otros componentes distintos. La tecnología de componentes va de la mano de la especialización.

Pero la especialización no fue inventada por la industria del automóvil. La Naturaleza la usa desde hace millones de años. Los seres vivos están constituidos por órganos, cada uno de los cuales es un conjunto de células altamente especializadas. La ventaja de este esquema es cada uno de ellos puede desenvolverse aislado de los otros.

Volviendo al ejemplo de la planta de ensamblaje de automóviles, las interfaces y especificaciones de todos sus componentes están acordadas de antemano para que no haya sorpresas durante el montaje. Dichas interfaces entre componentes se definen de la forma más simple posible para acelerar el tiempo de montaje.

En cualquier caso, en la industria del software tenemos otras peculiaridades: la replicación masiva nunca ha sido ningún problema. Nuestro principal objetivo es la personalización masiva: disponer de una amplia variedad de productos basados en un esqueleto común.

4 La Ley de Moore del Software

En esencia, un componente es una pieza de software que realiza una función bien definida y posee una interfaz bien definida. Claros ejemplos de los primeros componentes los tuvimos, por ejemplo, en los VBX introducidos por Visual Basic o los plug-ins para Photoshop. Fueron pasos importantes pero aún tenían dos defectos importantes por superar: sólo servían para un producto específico, limitando así su valor y eran concebidos como ampliaciones de la aplicación original, mientras que en las verdaderas aplicaciones basadas en componentes, estos constituyen la casi totalidad de la aplicación.

Las tecnologías mas recientes van mas allá: Actives, Java Beans, componentes desarrollado con herramientas que cumplen las especificaciones CORBA, SAP, etc. Estas tecnologías potencian los dos principales beneficios de la tecnología de componentes:

- La división en componentes reduce la complejidad, permite la reutilización y acelera el proceso de ensamblaje de software.
- Los creadores de componentes pueden especializarse creando objetos cada vez mas complejos y de mayor calidad.
- La interoperabilidad entre componentes de distintos fabricantes aumenta la competencia, reduce los costes y facilita la construcción de standards. El software se hace cada vez mas rápido, de mejor calidad y a menor coste.
- La principal implicación dentro de la industria del software es que esta se dividirá en dos: fabricantes y ensambladores de componentes.

5 El Secreto está... en el Empaquetado

Si los componentes existen desde principios de los años 90 ¿por qué ahora suponen una revolución? Porque los beneficios son alcanzables sólo ahora que la tecnologías para empaquetarlos ha alcanzado la suficiente madurez. Actives y Java Beans son dos

buenos ejemplos: ambos proporcionan los contenedores donde depositar el código de forma que podamos manejarlo sin preocuparnos del lenguaje en el que están escritos.

El empaquetado del código ha tardado tanto en desarrollarse porque va en contra de la norma de la mayoría de los programadores que persiguen la eficiencia del código por encima de la eficiencia en el desarrollo. En los principios de la informática, las máquinas eran caras y los programadores baratos, de forma que estos eran aleccionados para conseguir los mejores rendimientos con sus desarrollos. La idea de colocar capas de código innecesario con el único propósito de facilitar el desarrollo de aplicaciones parecía impensable.

Hoy en día, por el contrario, las máquinas son baratas y la gente que sabe trabajar con ellas muy cara. Entonces aparecieron las técnicas orientadas a objetos. Colocar datos y funciones juntas formando objetos fue un gran paso sin el que ninguna de las tecnologías de empaquetado de componentes actuales hubiera prosperado. La orientación a Objetos es el mayor paradigma que jamás ha existido en el mundo de la informática.

No obstante, los primeros intentos de empaquetado de código fueron un fracaso: los desarrolladores no podían creer que el envoltorio fuese más complejo que el código que contenía. Hoy en día, el empaquetado de código sigue siendo complejo y arrastra considerables problemas: aumenta considerablemente el tamaño de los programas, empeora el rendimiento de estos y hace consumir más recursos de las máquinas donde se ejecutan. Exactamente los mismos problemas que se les atribuyen a las GUI y a pesar de ello prácticamente todo el mundo usa alguna.

A pesar de estos problemas, las ventajas acabaron superando a los inconvenientes y aunque las tecnologías de empaquetado de código siguen siendo muy complejas de construir, han alcanzado un alto grado de facilidad de uso, de forma que los desarrolladores dedicados a la fabricación de componentes sólo tienen que ocuparse de desarrollar cada vez mejores componentes sin preocuparse de nada más.

A pesar de que hoy existen diversas técnicas de empaquetado de componentes, con grandes diferencias según de donde procedan, también tienen importantes similitudes:

- **Transparencia** en cuanto a la localización, permitiéndonos usar un componente sin tener que preocuparnos de donde se encuentra físicamente, incluso si este es cambiado de sitio.

- **Definición de la interfaz.** Especifican la interfaz que el componente proporciona de forma independiente al lenguaje de programación usado o el sistema operativo donde vamos a usarlo. Normalmente se inspiran en la sintaxis usada por las DLL, muy similar a la usada en las llamadas a funciones en C++
- **Invocación.** Soporte para cargar y ejecutar los componentes cuando sean necesarios, enviando y recibiendo eventos y comunicando con otros componentes y el resto de la aplicación a través de la red.
- **Introspección.** Una forma de aislar el componente del mundo exterior, centrándonos solamente en lo que hace y como lo hace.
- **Distribución.** Los componentes pueden transmitirse a través de una red.

La explosión de la tecnología de componentes ha sido lenta, en parte debido al sentimiento de insatisfacción que la Orientación a Objetos dejó en sus inicios: las promesas de facilidad en la reutilización de código nunca fueron cumplidas. Hasta ahora.

6 Creando la Arquitectura de una Empresa

Diseñar la arquitectura de sistemas de una gran organización es una tarea que puede resultarle intimidatoria a mucha gente. Son tres los requisitos para empezar: un departamento de arquitectura, la redacción de un anteproyecto de diseño de la arquitectura y un documento que recoja los valores que intentamos impulsar con ella.

Es fácil de olvidar, dado el alcance actual, que el uso de la informática en las organizaciones es un fenómeno relativamente reciente. No es de extrañar, pues, que las estructuras y procesos para obtener un valor real de esta inversión hayan retrasado la inversión en si misma. El resultado ha sido un gigantesco enredo.

La tónica seguida generalizadamente hasta el momento por la mayoría de las empresas ha sido buscar la robustez de los standards, comprando toda su tecnología a un único fabricante o exigiendo el desarrollo de aplicaciones que funcionen sobre una rígida arquitectura.

Lo primero es crear nuestra dirección de arquitectura (que puede estar formada por un equipo de personas o alguien a tiempo parcial, según el volumen de nuestra organización) que coordine y se asegure de que la arquitectura de nuestro sistema facilita a las aplicaciones existentes (y futuras) la consecución de los objetivos de nuestra em-

presa. Las tareas del departamento de arquitectura implican una amplia visión de la actividad de la organización, estar al día de los mas relevantes progresos tecnológicos y asegurarse de que los equipos que trabajan en el desarrollo de aplicaciones cumplen las directivas oportunas.

El jefe del departamento de arquitectura debe de jugar entre el punto de vista del empresario y el del técnico. Debe de ser pragmático antes que idealista. Evangelista antes que dictador. Y debe, por supuesto, ser de la absoluta confianza del director de sistemas de la organización.

Para comprender donde no se debe de meter el departamento de arquitectura, debemos de volver al modelo de cuatro capas de nuestra organización: los desarrolladores son sus clientes y los responsables de la infraestructura sus proveedores. Los arquitectos de sistemas que pasan demasiado tiempo eligiendo sistemas operativos o rediseñando las líneas de negocio de su organización no invierten el tiempo necesario en su trabajo.

De cualquier modo, la línea de división entre infraestructura y arquitectura puede ser muy difícil de ver: muchos de los elementos que en un momento dado forman parte de la arquitectura, posteriormente se consideran parte de la infraestructura, como ha ocurrido con las redes locales y ocurrirá con los sistemas de mensajería. En cualquier caso, si es responsabilidad del jefe de arquitectura promover este tipo de progresos porque, como es lógico, con una débil infraestructura no es posible edificar una sólida arquitectura.

Una importante clave en el enfoque moderno de la informática es ver las aplicaciones como una colección de servicios. Estos servicios deberían de ser, hasta donde sea posible, independientes unos de otros, de forma que podamos sustituir, eliminar o añadir nuevas funcionalidades sin demasiado esfuerzo.

Separar de esta forma nuestros servicios nos proporciona un valor añadido: poder elegir la mejor tecnología para cada uno de ellos: los datos de los clientes en una base de datos relacional, los de los empleados en un directorio que cumpla las especificaciones LDAP y las descripciones de nuestros productos en un servidor web.

El problema en este idílico paisaje es que la mayoría de los servicios con las que ya cuentas en tu organización no están correctamente empaquetadas y con sus interfaces bien definidas, sino que se encuentran sepultados bajo aplicaciones existentes en paquetes cerrados de sistemas propietarios. Por esto es necesario un anteproyecto del

sistema que deseamos tener., de forma que podamos trazar un camino para llegar hasta el. En este documento deben de aparecer los servicios que tenemos y las aplicaciones que los usan Podría señalar también los puntos donde esos servicios ya existen y donde es preciso crearlos. Cualquier nuevo diseño debe de responder a dos preguntas: de que existentes servicios puedo hacer uso y a que nuevo servicio puedo yo contribuir. Esto supone no volver a pensar en aplicaciones aisladas para comenzar a pensar en términos de compartir servicios de aplicaciones.

El último punto que necesitamos es crear un documento con los valores de nuestra arquitectura. Debe de ser un documento breve, de menos de 1000 palabras, distribuido a todo el personal de sistemas de la empresa. Debe de cubrir puntos como definir cuando debemos usar dos capas y cuando tres, que middleware usaremos y que standards (sistemas operativos, de mensajería, etc.) no están abiertos a debate. Debe de ser un documento deliberadamente minimalista, permitiendo libertad a la gente con creatividad para elegir cual es la mejor solución para resolver su problema particular.

Tenemos que concentrarnos en tres ventajas fundamentales que este documento debe de aportarnos. En primer lugar elimina los debates estériles señalando claramente que puntos no están abiertos a debate, de forma que el personal se centre en debatir los problemas reales. En segundo lugar nos ayuda a identificar al personal que cumple los mejores requisitos para trabajar con nuestra arquitectura, de forma que se puede utilizar este documento como criterio de selección de los nuevos técnicos. En tercer lugar, le proporciona a los técnicos la posibilidad de enfocar sus esfuerzos y su creatividad en campos en los que realmente serán apreciados.

A continuación se muestra un ejemplo muy simple a partir del cual se podría empezar a trabajar:

Este documento, que muestra nuestra posición frente a los mas relevantes puntos de la arquitectura de sistemas de esta empresa, fue redactado por [el jefe de arquitectura] y ratificado por [el director de sistemas] Fue revisado pro ultima vez en [fecha] y si ha transcurrido mas de un año, probablemente tienes en tus manos una versión obsoleta.

Este documento se ha redactado con la idea de facilitar el trabajo a todo el personal de sistemas de [nombre de la organización] y no para censurarle ni limitarle. Si tienes una razón lo suficientemente fuerte como para contravenir alguno de los puntos aquí indicados, documéntala y háznosla llegar.

1. Nunca olvides que la actividad de nuestra empresa está encaminada a [actividad de la empresa] y que en el departamento de sistemas, nuestra principal prioridad es desarrollar sistemas que soporten esa línea de negocios.

Dejemos a otros la innovación en la tecnología para que nosotros innovemos en como aplicar esa tecnología en una aplicación real.

2. Debemos maximizar el valor de nuestro trabajo desarrollando servicios para aplicaciones en lugar de aplicaciones aisladas, permitiéndonos reutilizar en el futuro esos servicios para crear nuevas aplicaciones rápida y fácilmente. Nuestro anteproyecto de servicio de aplicaciones define los servicios que necesitamos y si existen o no en este momento. Cuando diseñes nuevas aplicaciones te pedimos que: uses todos los servicios existentes que necesites y consideres a que nuevos servicios puede contribuir tu aplicación.
3. En el nivel mas bajo, pensamos que todas las aplicaciones deben de estar ensambladas por componentes software. La tecnología que uses para ello no es tan importante, pero a menos que tengas una buena razón para no hacerlo preferimos que uses [tecnología de componentes preferida, por ejemplo ActiveX] porque [motivos para preferirla, por ejemplo, parece que dominará el mercado durante algunos años].
4. Deseamos que las herramientas y tecnologías basadas en Web sean standards en nuestra empresa de aquí a dos años. Considera esto y realiza una aproximación siempre que sea posible.
5. Usa tres capas para todas las aplicaciones no triviales (por ejemplo, las entradas simples de datos podrían ir sobre dos capas). Esto facilita el mantenimiento y la reutilización del código y mejora el rendimiento de las aplicaciones.
6. Para aplicaciones basadas en objetos distribuidos, preferimos [nombre de la tecnología, por ejemplo CORBA]. Valoramos el tiempo de desarrollo por encima del purismo.
7. Excepto para algunos informes especiales, no necesitamos recoger datos de ningún tipo. Existe una estrategia especial de Data Warehouse que se encarga de ello.
8. Si puedes encontrar un paquete desarrollado que realice el 80% de la funcionalidad que buscamos úsalo, en otro caso considera un desarrollo a medida: la excesiva personalización de un paquete conlleva demasiados riesgos. Los paquetes que no son lo suficientemente abiertos para jugar su papel de colaboración en nuestro anteproyecto de servicios de aplicaciones deberían de ser rechazados.
9. Esperamos que la comunicación entre aplicaciones se realice mediante RPC y productos de mensajería.
10. Toma como inamovible la actual infraestructura. En ella incluimos Windows 98 y Office 2000. No desarrolles nada sobre entornos de 16 bits. Utiliza siempre entornos de 32 bits.
11. Considera la necesaria inversión en personal específico cuando investigues nuevas herramientas. Para la mayoría de las herramientas, esta inversión excede cualquier otra diferencia técnica las mismas. Esto significa que tenemos

una serie de preferencias: Oracle para bases de datos relacionales, SQL vía ODBC para el acceso a los datos, Microsoft NT como servidores, C++ o Java como herramientas de desarrollo de tercera generación y Visual Basic como herramienta de cuarta generación.

12. Evita reemplazar aplicaciones actuales a menos que sea totalmente indispensable.

7 Temas Importantes en la Arquitectura Moderna

Publicar un conjunto de valores para nuestra arquitectura requiere que estemos bien informados. El punto de vista de un jefe de arquitectura debe de contemplar los siguientes puntos:

7.1 Estándares

El alza de Internet y todas las tecnologías que van con ella está revolucionando el mundo de la informática. Algunos de sus efectos son rápidamente visibles, pero otros no lo son tanto. El jefe de arquitectura debe de estar bien informado de los estándares emergentes

En algunos casos nos encontraremos con un serio problema a la hora de tomar partido por dos estándares que, aparentemente, poseen la misma funcionalidad, por ejemplo DCOM y CORBA. La decisión debe de tomarse recopilando información claramente imparcial y, si no es posible o no es lo suficientemente aclaratoria, probándolos por nosotros mismos. Lo peor que podemos hacer es no usar ninguno de los dos.

7.2 La capa intermedia

Ya hemos visto que uno de los grandes progresos de la arquitectura moderna ha sido la subdivisión del software. El 'pegamento' que une estas partes formando una aplicación completa es lo que se conoce como middleware. Imaginemos dos programas (A y B) corriendo separadamente cada uno en una máquina. A llama a B con un problema. B trabaja en la resolución del mismo y cuando acaba devuelve la respuesta a A. El middleware es el que nos permite esta funcionalidad, pero existen algunos problemas derivados de esta forma de actuación ¿qué debería de hacer el proceso A mientras que B trabaja en la resolución de su problema? ¿esperar? ¿Cómo puede B comunicarle a A algo a menos que este se lo requiera? ¿Cómo puede B saber

donde se encuentra A?;Qué hace A si B se cae?;Si B es reemplazado?;Si cientos de A's requieren una respuesta de una única instancia de B? Como vemos, los middlewares deben de resolver complejas situaciones pero sin añadir excesiva complejidad a la arquitectura.

Existen dos tipos esenciales de middlewares: los basados en RPC (Remote Procedure Calls) y los basados en MOM (Message Oriented Middleware). Middlewares basados en MOM son inherentemente asíncronos y lo mas difícil en ellos es definir correctamente la estructura de los mensajes. Los middlewares basados en RPC son mas sencillos de usar, puesto que la sintaxis de las llamadas es prácticamente idéntica a la de una llamada en C, pero el rendimiento de estos sistemas suele ser mas pobre. Existen algunos fabricantes que distribuyen productos capaces de funcionar en uno u otro modo.

La mejores herramientas hoy en día están basadas en CORBA, están orientadas a mensajes y tienen como inconveniente que son mas difíciles de usar por los programadores. Las herramientas de Microsoft basadas en COM+ son mas limitadas pero muy sencillas de usar y son recomendables si anteponeamos esta característica a la longevidad y la calidad del producto.

7.3 Estructura cliente / servidor a dos o tres capas

La estructura cliente / servidor a dos capas nació el día en que alguien conecto su PC a una máquina UNIX. A los usuarios les gusta la facilidad de uso de los PC's y a los administradores la seguridad que les reporta un servidor UNIX. Nadie lo llamó entonces arquitectura a dos capas porque no existía ninguna mas. La novedad de contar con una interfaz gráfica en lugar de una pantalla verde en modo texto fue bien recibida. Los desarrolladores comenzaron entonces a enriquecer sus productos con nuevas funcionalidades a sus productos gracias a las mejores herramientas de desarrollo existentes para los PC's. Los clientes 'engordaron' haciéndose mas pesados y lentos. La alternativa, meter parte de esta nueva funcionalidad en el back-end no era demasiado atractiva, así que se buscó la solución introduciendo una tercera capa central situada entre el cliente y el servidor para sostener gran parte del peso de esta nueva funcionalidad.

Pese a sus evidentes ventajas, los sistemas en tres capas han tardado en generalizarse debido a que son mucho mas caros y difíciles de desarrollar. La arquitectura

cliente / servidor a dos capas sigue siendo útil para la mayoría de los casos y ahora aparece la tecnología web para acercar la arquitectura a tres capas a las masas.

7.4 La web

La tecnología web ha cambiado sustancialmente la forma en que las aplicaciones son desarrolladas. Si nada lo para, en muy poco tiempo el standard para crear interfaces gráficas será el HTML. Pero la web tiene muchas mas cosas que ofrecer que meramente la apariencia externa. Internet pronto conectará a la totalidad de los ordenadores del planeta. Esto cambiará por completo las reglas del desarrollo de aplicaciones. Las organizaciones que permanezcan aisladas no podrán beneficiarse de estas grandes ventajas: Tu, tus clientes (actuales y potenciales), tus empleados y tus proveedores estarán continuamente conectados.

La web ha popularizado también un gran conjunto de tecnologías actuales y pasadas (HTML, TCP/IP, Java, etc. Los arquitectos tenemos ahora un mayor conjunto de herramientas con las que jugar.

Veamos un caso típico: una herramienta cliente podría consistir simplemente en una amistosa interfaz html utilizable mediante nuestro navegador favorito. Este cliente se conecta a través de un servidor web con una capa intermedia formada por componentes escritos en diversos lenguajes (C++, Visual Basic y Java) y empaquetados con Actives o Java Beans. Todo ello se sustenta sobre un servidor de transacciones (como Microsoft MTS o TP Tuxedo) y se conecta a una tercera capa de aplicaciones propietarias (mediante Microsoft MQ o IBM MQSeries) y con un servidor de bases de datos vía ODBC. Todo ello permanece unido mediante un lenguaje basado en scripts como Java Scripts o Visual Basic Scripts

Arquitecturas como esta popularizan por primera vez en la historia de la informática los beneficios de los lenguajes de componentes y la estructura a tres capas.

7.5 Empaquetado.

Utilizar tecnología de componentes empaquetados puede ahorrarte años y años de desarrollo. Ahorramos en costes de desarrollo y mantenimiento y nos beneficiamos además de la robusted que nos proporcionan componentes que han sido testeados por cientos de usuarios antes que nosotros.

Los problemas comienzan con la personalización. Habitualmente los paquetes de software no cumplen exactamente el 100% de nuestros requisitos. Podemos adaptarnos nosotros a ellos o, lo que parece más lógico, adaptarlos a ellos a nosotros.

Como regla general, si encontramos un paquete que cumple el 80% de nuestros requerimientos, debemos de comprarlo y personalizarlo. Si los requerimientos que cumple están por debajo de este porcentaje, es mucho mejor que desarrollemos el sistema por nosotros mismos. Las personalizaciones complejas pueden ser caras y, a menudo, inviables.

Donde mejor se comportan los paquetes es en los procesos que son prácticamente iguales en multitud de empresas, como la facturación, o cuando lo que queremos es personalizar el contenido y no la funcionalidad, como en sistemas de datos de gestión documental.

A veces ocurre que queremos sustituir un paquete pero parte de su funcionalidad es indispensable para otros elementos: el paquete se ha convertido en parte de la arquitectura. Para evitar estos indeseables efectos, debemos de envolver los paquetes y realizar la integración contra este envoltorio. Esto encarece el coste de desarrollo pero, a la larga, mejora la flexibilidad de nuestro sistema.

7.6 Envolviendo sistemas propietarios

Muchos sistemas propietarios nos proporcionan funcionalidades muy valiosas en determinadas misiones críticas. Son fruto de años y años de experiencia y su uso en una arquitectura bien diseñada puede ser muy beneficioso. La forma de conseguirlo es envolverlo con una interfaz que lo haga comportarse como uno de nuestros paquetes.

A pesar de usar tecnologías obsoletas, los sistemas propietarios suelen usar tres capas bien definidas: presentación lógica y almacenamiento de datos. La mejor forma de envolverla es acceder directamente a la lógica de la aplicación. Para ello la aplicación debe de proporcionarnos un API.

Si esto no es posible, la segunda opción sería acceder directamente a los datos almacenados. Esto supone un perfecto conocimiento de la estructura de almacenamien-

to de la aplicación y, a menudo, tener que solventar problemas de sincronización y gestión de bloqueos.

La última línea de ataque sería atacar el nivel de presentación. El envoltorio se comunica con la aplicación igual que lo haría un usuario. Tecleando datos y recogiendo respuestas de la salida pro defecto por medio de una interfaz de terminal.

Las tres técnicas son caras y difíciles, pero seguramente usar la aplicación será mucho mejor que desecharla o desarrollar otra similar.