TÍTULO: Modelo para el análisis de arquitecturas de intermediación electrónica
AUTOR: Francisco Valera Pintor
DIRECTOR: José Ignacio Moreno Novella

RESUMEN DE LA TESIS

La tecnología asociada al mundo del comercio electrónico ha visto durante estos años cómo se pasaba de un período de incertidumbre inicial, a un período de tremendo optimismo en el que, coincidiendo con el auge de las '*empresas punto-com*', se desarrollaron rápidamente muchas tecnologías de todo tipo.

Actualmente, diferentes combinaciones tecnológicas han llevado a la construcción de arquitecturas, aplicaciones y plataformas con propósitos diversos que necesariamente han de convivir e interaccionar entre ellas. Por ese motivo, se aprecia la notable necesidad de desarrollar nuevas tecnologías y mecanismos orientados principalmente a salvar toda esta heterogeneidad existente.

Esta tesis parte del hecho de que la gran cantidad de tecnologías diferentes y las características tan particulares del entorno que las engloba, han elevado la complejidad del sector de forma muy considerable. Por un lado, a los desarrolladores de aplicaciones se les exigen unos conocimientos detallados sobre muchas tecnologías distintas. Por otro, muchas empresas se están viendo forzadas a cambiar sus procedimientos de selección tecnológica pues el clásico esquema de desarrollo a medida, empieza a no ser rentable. Además, el ritmo de aparición y evolución de tecnologías y arquitecturas sigue siendo alto y cada vez resulta más complicado analizarlas o clasificarlas.

Apoyándose en diversos criterios que surgen de la formalización del concepto de complejidad, se propone un modelo genérico para facilitar el análisis de las características de las arquitecturas de comercio electrónico centrándose principalmente en los aspectos tecnológicos de las mismas y haciendo especial hincapié en las plataformas de intermediación electrónica.

UNIVERSIDAD CARLOS III DE MADRID
Departamento de Ingeniería Telemática

Doctorado en Tecnologías de las Comunicaciones

TESIS DOCTORAL

Modelo para el análisis de arquitecturas de intermediación electrónica

Autor: Francisco Valera Pintor
Ingeniero de Telecomunicación

Director: José Ignacio Moreno Novella
Doctor Ingeniero de Telecomunicación

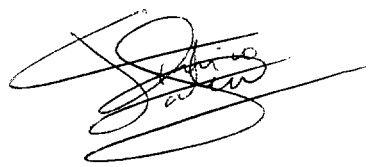
Leganés, abril de 2002

D. FRANCISCO VALERA PINTOR, con D. N. I. : 50852626 V

AUTORIZA:

A que su tesis doctoral con el título: **"Modelo para el análisis de arquitecturas de intermediación electrónica"** pueda ser utilizada para fines de investigación por parte de la Universidad Carlos III de Madrid.

Leganés, 14 de junio de 2002



Fdo.: Francisco Valera Pintor

Título: Modelo para el análisis de arquitecturas de intermediación electrónica
Autor: Francisco Valera Pintor
Director: José Ignacio Moreno Novella

TRIBUNAL

Presidente: Julio Berrocal Colmenarejo

Vocales: Carlos Delgado Kloos
Jaime Mª Delgado Mera
Víctor Villagrá González

Secretario: Arturo Azcorra Saloña

Realizado el acto de defensa y lectura de la Tesis el día 14 de Junio de 2002 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de Sobresaliente Cum Laude (por unanimidad)

LOS VOCALES

C. Delgado Kloos

Jaime Delgado

Víctor A. Villagrá

EL SECRETARIO

EL PRESIDENTE

A mis padres y a mi hermana
A Rosa

Agradecimientos

Son muchas las personas que con sus ánimos, consejos, amistad o cariño, han contribuido a que el trabajo realizado a lo largo de estos últimos años fructificase finalmente en esta tesis. Por eso me gustaría dejar aquí constancia de mi más sincero agradecimiento a todas ellas.

A José Ignacio Moreno, director de esta tesis. Gracias por tus consejos, por compartir conmigo tu experiencia, por tu constante apoyo profesional y sobre todo personal y por la confianza que siempre has depositado en mí.

Al resto de compañeros del Departamento de Ingeniería Telemática de la Universidad Carlos III de Madrid, especialmente a Alberto, David, Arturo, Ignacio, María y Marcelo, por vuestras muestras de interés, vuestros consejos, vuestros ánimos y por todas las ofertas de ayuda que he recibido. Gracias también a Ricardo, por tu ayuda con las clases estos últimos meses, que me han dado el tiempo que necesitaba para acabar este trabajo.

A los compañeros del grupo de redes del Departamento de Ingeniería de Sistemas Telemáticos de la Universidad Politécnica de Madrid con los que tuve la oportunidad de iniciar mi carrera profesional y con los que he continuado recorriendo el camino que ha culminado con esta tesis. Gracias a Jorge, Publio y Rodrigo por tantas horas juntos en los primeros proyectos y a Julio, Víctor, Juan Ignacio, Luis y Enrique por enseñarme con vuestra experiencia y por las pruebas de afecto y de confianza que siempre me habéis mostrado.

A mis buenos amigos Jose y Cristina, porque a través de los años siempre me habéis seguido de cerca, demostrándome un interés y un cariño muy personal. Gracias por vuestro empuje, vuestra amistad y vuestra constancia.

Al equipo de Judo y la gente de Tai-Chi, especialmente a Luis, Antonio y Gonzalo, y a Marcos, porque me ayudáis a darme cuenta de que a lo que hago durante mi trabajo semanal, sólo hay que darle la importancia que tiene.

A Chelo, Noe, Amaia y Bea por preocuparse por mí y no dejar pasar nunca una oportunidad de preguntar sobre el estado de la tesis y más importante aún, sobre cómo estaba yo.

A mis abuelos Paco, Carmen y Adolfo, que siempre se interesaron con cariño por todo lo que yo hacía.

A Mamanana que nunca deja de preguntar por cuanto y cuantos me rodea.

A mis padres y mi hermana. Gracias por vuestro cariño y apoyo incondicional, por preocuparos a todas horas y por estar siempre detrás de mí, en cada paso que doy.

A Rosa. Gracias por estar conmigo en todo momento y por aguantar la constante dedicación al trabajo y a la tesis que he sostenido estos últimos años. Gracias por recordarme siempre lo que realmente es importante. Sin ti esta tesis ni estaría escrita ni tendría sentido.

Resumen

La tecnología asociada al mundo del comercio electrónico ha visto en estos últimos años cómo se pasaba de un período de incertidumbre inicial, en el que no estaba claro lo que dicha tecnología podía dar de sí, a un período de tremendo optimismo en el que, coincidiendo con el auge de las denominadas '*empresas punto-com*', se desarrollaron multitud de tecnologías de todo tipo con gran rapidez.

Actualmente, diferentes combinaciones de esas tecnologías han llevado a la construcción de arquitecturas, aplicaciones y plataformas con propósitos diversos que necesariamente han de convivir e interaccionar unas con otras. Por ese motivo, se aprecia ahora la notable necesidad de desarrollar nuevas tecnologías y mecanismos orientados principalmente a salvar toda esta heterogeneidad existente.

Esta tesis parte del hecho de que esta gran cantidad de tecnologías diferentes y las características tan particulares del entorno que las engloba, han elevado la complejidad del sector de forma muy considerable. Por un lado, a los desarrolladores de aplicaciones se les exigen unos conocimientos muy detallados sobre muchas tecnologías distintas. Por otro, las empresas se están viendo forzadas en muchos casos a cambiar sus procedimientos de selección tecnológica pues el clásico esquema de desarrollo a medida, está empezando a no ser rentable. Además, el ritmo de aparición y evolución de tecnologías y arquitecturas sigue siendo muy alto y cada vez resulta más complicado analizarlas o clasificarlas.

Apoyándose en diversos criterios que surgen de una formalización del concepto de complejidad, esta tesis propone un modelo genérico para facilitar el análisis de las características de las arquitecturas de comercio electrónico centrándose principalmente en los aspectos tecnológicos de las mismas y haciendo un especial hincapié en las posibilidades de las plataformas de intermediación electrónica.

Palabras clave: arquitectura, comercio electrónico, complejidad, comunicaciones, distribución, objetos, intermediación

Abstract

Electronic Commerce related technologies have been passing during these last years through an uncertain period, in which their market possibilities were not really clear and later through an optimistic period where, at the same time that the so called '*dot-com companies*' became popular, a lot of new technologies began to appear at a very high rate.

Nowadays, different combinations of those technologies have lead to the development of architectures, applications and platforms with diverse objectives, which must inevitably interact and communicate to each other. Because of this, there is now a notorious necessity about the development of new technologies and mechanisms in order to solve all this existing heterogeneity.

This thesis is based on the fact that this great number of different technologies and the particular properties of the environment where they are deployed, have considerably increased the complexity associated to electronic commerce. Developers are now asked for a much more specific knowledge about many more technologies and enterprises are being obliged to change their technology selection process, because the option of building applications instead of buying them is not worthwhile anymore. In addition, technologies and architectures appearance rate is still very high and their analysis or classification is becoming more and more difficult.

Based on the different criteria emerging from the formalization of the concept of complexity, this thesis proposes a generic model in order to facilitate the analysis of electronic commerce architectures characteristics, mainly focusing on technological issues and paying special attention to electronic brokering possibilities.

Keywords: architecture, electronic commerce, complexity, communications, distribution, object, brokering.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y objetivos

1.1 Introducción

En este capítulo se presenta el entorno general en el que se ha desarrollado la tesis, su principal motivación, sus objetivos y una breve reseña de cada capítulo para facilitar la lectura de la memoria.

Inicialmente se expone la situación actual de Internet y del comercio electrónico, así como las previsiones que se tienen relativas a sus respectivas líneas de evolución. Se comenta además el entorno socio-tecnológico que envuelve a las oscilaciones que el comercio electrónico ha ido experimentando en los últimos años. Todas estas tendencias son muy importantes desde el punto de vista de la tesis, pues los movimientos tecnológicos y la creciente complicación de los mismos, que es lo que aquí se analiza, tienen una estrecha vinculación con ellas.

En la tesis, el comercio electrónico se particulariza en las llamadas plataformas de intermediación electrónica, que no son sino un modelo de negocio especialmente potente e innovador del comercio electrónico genérico. Pese a que en capítulos posteriores (ver capítulo 5) se examina el tema con mayor detalle, se introduce ya en este capítulo por su especial relevancia en el desarrollo de la tesis.

En este marco se presenta posteriormente la principal motivación de la tesis: la tremenda complejidad que está adquiriendo la tecnología que rodea el mundo del comercio electrónico y la problemática que todo ello lleva asociada. Dicha motivación se

abordará en el capítulo 6, proponiendo un modelo de complejidad para el análisis de plataformas de intermediación electrónica.

Por último se especifican los objetivos concretos que se persiguen, orientados a manejar esa complejidad que se comenta y cuyo concepto se formalizará convenientemente más adelante.

1.2 Internet y el comercio electrónico

Actualmente no cabe ya ninguna duda sobre el tremendo impacto que están teniendo las Tecnologías de la Información y en particular Internet en la sociedad a nivel mundial.

Las previsiones más optimistas sobre el crecimiento del número de usuarios conectados a la red se han visto cumplidas con creces y aún se prevé un crecimiento sostenido durante varios años más (Figura 1).

En Europa el número de usuarios ha superado recientemente (año 2001) al de Estados Unidos y concretamente en España, se estima que el 20% de los hogares tienen ya acceso a Internet (aunque se está todavía muy por debajo del 50% de algunos países nórdicos, las encuestas indican que cerca de un 40% de los que todavía no son usuarios en España, tienen intención de serlo en un futuro próximo [AUI]).

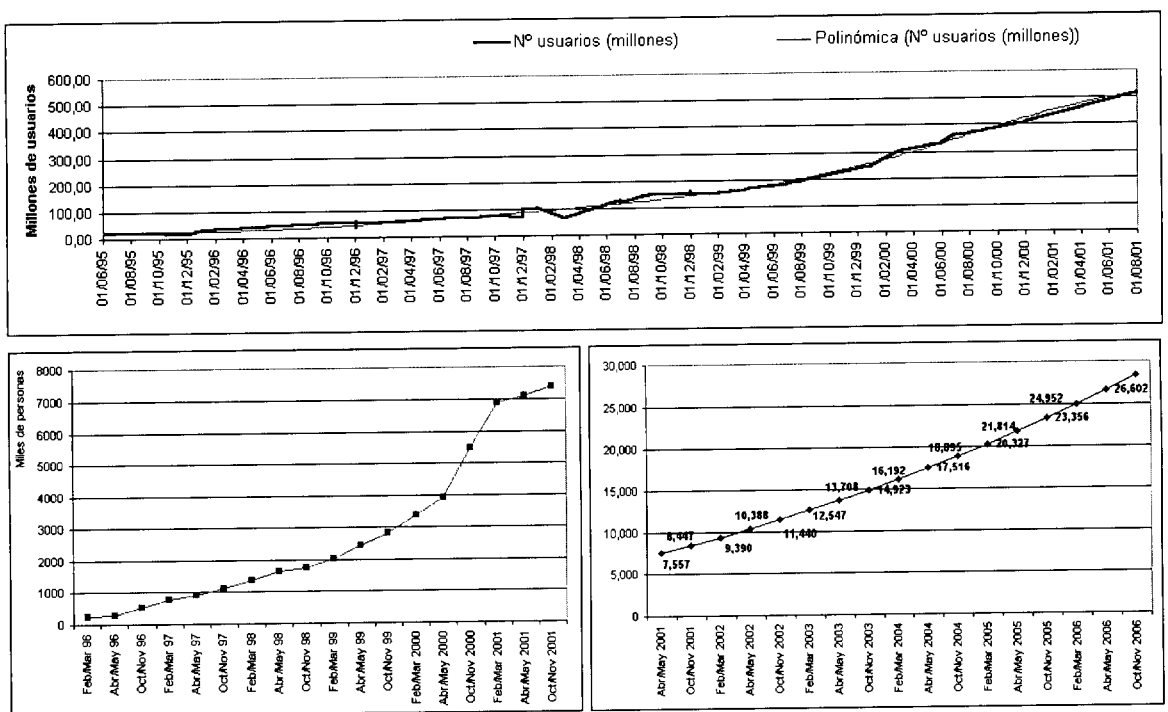


Figura 1. Número de usuarios de Internet (superior) y evolución y previsión del número de usuarios en España (inferior) [AUI]

Todas estas previsiones son las que hace unos años fomentaron la progresiva incorporación a Internet, una red que en principio surgió y evolucionó con fines meramente divulgativos en el ámbito académico y de investigación, de diferentes

servicios comerciales y que ha llevado a lo que hoy en día se conoce como comercio electrónico: 'hacer negocios por Internet' [Büs97].

La idea de comercio electrónico, no es ni mucho menos algo nuevo. Durante bastantes años, las empresas han estado intercambiando datos de negocio a través de diferentes redes de comunicaciones e incluso se buscaron mecanismos más formales como EDI (*Electronic Document Interchange*, [UN/EDIFACT]) para facilitar esta tarea. Sin embargo, todo ese negocio estaba en general restringido a redes propietarias y era una actividad que se llevaba a cabo exclusivamente entre empresas (comercio B2B, *Business to Business*).

Actualmente el rápido crecimiento de Internet, tanto en tamaño como en posibilidades, ha hecho del comercio electrónico algo global, incorporando al sector a multitud de nuevas empresas y a diferentes tipos de usuarios finales.

El comercio electrónico tal y como se entiende ahora, incluye por lo tanto, además del tradicional B2B, actividades de tipo B2C (*Business to Consumer*), actividades B2A (*Business to Administration*) y numerosas combinaciones más, como ya se verá.

En cualquier caso y pese a la vertiginosa expansión que ha experimentado el comercio electrónico, a nivel de usuario todavía no ha alcanzado un nivel de penetración que se pudiese catalogar de importante (los servicios que más se utilizan son el Web y el correo electrónico, seguidos de cerca por la descarga de aplicaciones y las charlas interactivas [AIMC]).

En nuestro país, sí hay algunos comercios que llevan tiempo ofreciendo productos por Internet (si no venta sí al menos publicidad sobre sus productos o catálogos) y numerosos bancos ofreciendo prestaciones *on-line*, pero lo cierto es que son en general servicios de reciente implantación y aún no demasiado difundidos a nivel popular.

En este sentido, son alentadoras las últimas encuestas del EGM [AIMC], que revelan no sólo la ya mencionada creciente incorporación de usuarios (habituales) a Internet, sino el hecho de que más del 50% de esos usuarios reconocieron haber tomado una decisión de compra de productos o servicios, orientados, motivados o informados por contenidos del Web y cerca del 48% efectivamente han realizado alguna compra a lo largo del año 2001 (ver Figura 2).

De la misma forma la mitad de los usuarios españoles declararon haber utilizado la banca electrónica (aunque sólo el 25% realizó transacciones) y otros servicios como las subastas electrónicas (si bien estas presentan mucho menor nivel de utilización con tan sólo el 10,6%).

Esto quiere decir que, independientemente de que la utilización de servicios de comercio electrónico por Internet esté empezando a crecer ahora en nuestro país (y de hecho, aunque quizá podamos ir un poco rezagados, la tendencia es similar en todo el mundo), sí se observa ya una importante utilización de la red para obtener información de productos y servicios.

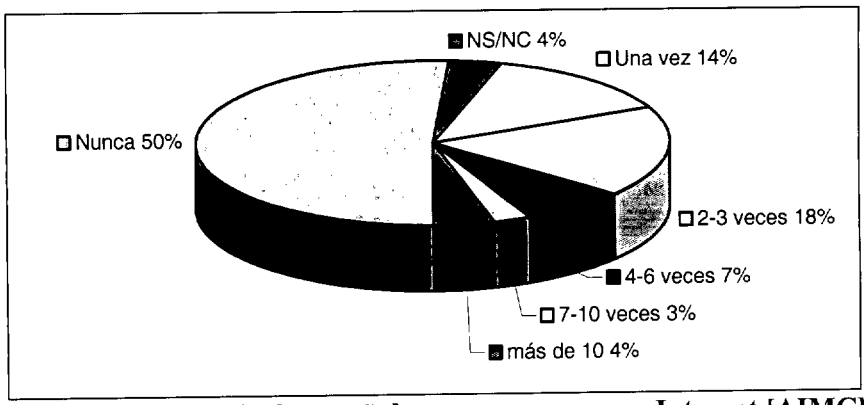


Figura 2. Porcentaje de españoles que compran por Internet [AIMC]

En cuanto a la calidad del servicio y al grado de satisfacción alcanzado por el servicio, las encuestas realizadas a lo largo del año 2001 demuestran que en general, los usuarios que realizan sus compras por Internet están muy satisfechos del servicio (según [KSR] un 70% de los usuarios se declararon muy satisfechos del servicio, cifra que llega al 88,5% en el caso de los internautas españoles, [AIMC]). También es verdad que los productos adquiridos (en la mayoría de los casos libros, música, software y viajes), no ofrecen tantos problemas como se pueden plantear en la compra de otro tipo de productos más delicados.

En contraposición a estos datos, hay que destacar que pese a que ahora durante los años 2000 y 2001 casi se llega al 50% de encuestados que declaró haber comprado algo por Internet y en 1999 se tenían menos de un 35% de respuestas afirmativas, no es menos cierto que la mayoría de los usuarios sólo compró de una a tres veces durante un año (Figura 2).

Los motivos que los usuarios apuntan como más importantes para no comprar en Internet son entre otros la falta de información, la desconfianza en los medios de pago, el miedo a dar los datos personales, desconfianza en la entrega de los productos, la lentitud de las conexiones o los costes del envío. Salvo este último factor, el resto de los puntos cada vez se percibe de forma más optimista y de hecho entre los usuarios españoles, prácticamente se considera ya tan seguro pagar con tarjeta de crédito en un restaurante como dar el número de la tarjeta por Internet [AUI].

Por otro lado, hay que considerar también, que además del típico comercio B2C basado en el modelo de la tienda virtual, que es básicamente el que se ha estado comentando en este apartado y al que suelen referirse las encuestas, se están explorando nuevos modelos de negocio que facilitarán la evolución del comercio electrónico.

1.3 Situación actual del comercio electrónico

Independientemente de lo que las cifras anteriores puedan indicar, es bien sabido que durante estos últimos años el comercio electrónico ha estado sometido a notables altibajos (Figura 3).

Hace cuatro o cinco años la percepción social del comercio electrónico era absolutamente optimista y el florecimiento de las llamadas empresas *punto-com* hacía que este tipo de negocio estuviese en pleno apogeo. Tanto era así que incluso se creó una cotización bursátil alternativa para el sector y prácticamente todos los días se podían leer noticias sobre la aparición de nuevas empresas o el impresionante crecimiento de las empresas existentes.

Progresivamente el panorama económico y comercial fue cambiando y actualmente el hundimiento que se ha apreciado en las *empresas Internet*, arrastrando hasta la quiebra a muchas de ellas, ha llevado a pensar en un hundimiento paralelo de todo el sector asociado a la denominada nueva economía y a una ralentización en el desarrollo del comercio y negocio electrónico en general.

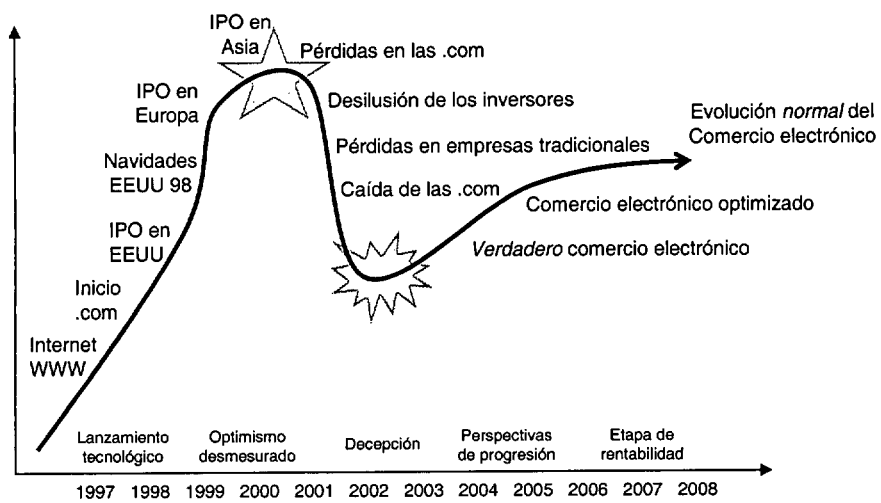


Figura 3. Gráfica de evolución del comercio electrónico [Garner]

Afortunadamente, hay datos que avalan la teoría de que esta sensación no concuerda exactamente con la realidad [Gan01]: incluso sufriendo el paso atrás motivado por la ralentización de la economía a nivel global y la caída del sector tecnológico de las *punto-com*, las empresas han incrementado entre un 20 y un 30% sus inversiones en negocio electrónico, esperando todavía aumentar sus ingresos más del 50%.

Siguiendo con esta línea de razonamiento, un estudio realizado en 2001 por la consultora IDC [IDC] en más de 25 países a decenas de miles de empresas, pone también de manifiesto este hecho de que las percepciones acerca de la evolución del comercio electrónico distan en ocasiones de lo que realmente arrojan las cifras.

Frente al convencimiento que se observa sobre el hecho de que el negocio electrónico está parado debido a la desaparición de las *punto-com*, que el comercio B2C está en clara decadencia, que el negocio electrónico realmente sólo es para empresas grandes, que el negocio electrónico sólo tiene que ver con el Web o que los mercados electrónicos (*eMarketplaces*) son una utopía de hace 5 años, el estudio avala con datos la situación real del sector.

Una de las claves, está en el hecho de que a las empresas de *origen tradicional* la crisis de las *punto-com* no les ha afectado demasiado y más del 90% del movimiento económico del sector viene precisamente de ellas, del comercio B2B (ver Figura 4 y Figura 6).

Estas empresas ya hacían negocios electrónicos antes de la explosión de las empresas virtuales y su base comercial si bien se ha ido adaptando progresivamente al entorno de Internet, en general no ha quedado totalmente comprometida a este medio.

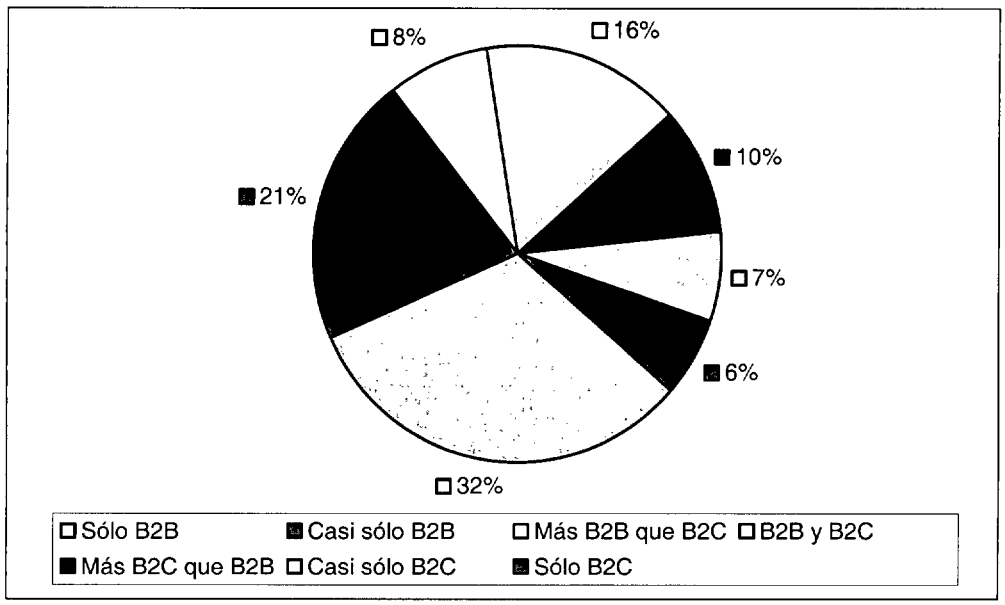


Figura 4. Distribución de la orientación comercial de las empresas en 2001 [KSR]

Dichas empresas continúan apostando por la investigación en el Web y en la venta en línea y siguen apreciando las diferentes ventajas que ofrece el sector como la baja barrera de entrada, la fácil integración con sistemas de respaldo, los ahorros económicos que se obtienen por el hecho de integrar el comercio electrónico en la cadena de valores, etc. Tal y como se aprecia en la Figura 5, actualmente hay más de 12 millones de empresas con presencia en el Web y se espera que para 2005 se haya duplicado esta cifra.

Con respecto a la distribución por tamaño de empresas, de los 12 millones de sitios Web que había a final de 2001, pese a que las empresas pequeñas que tienen Web no llegan al 91% que alcanzan las grandes empresas, sí hay ya casi un 60% de PYMES con Web. El abaratamiento de los costes de conexión y mantenimiento de un sitio Web y las ventajas que ofrece el hecho de acceder a un mercado potencialmente muy grande, hacen que la previsión para dentro de cinco años, eleve a un más del 80% la presencia en el Web de las empresas pequeñas.

Es también interesante el dato de que aunque el comercio B2B es sin duda mayor (en ingresos) que el B2C, desde el punto de vista de los sitios Web, la situación está pareja.

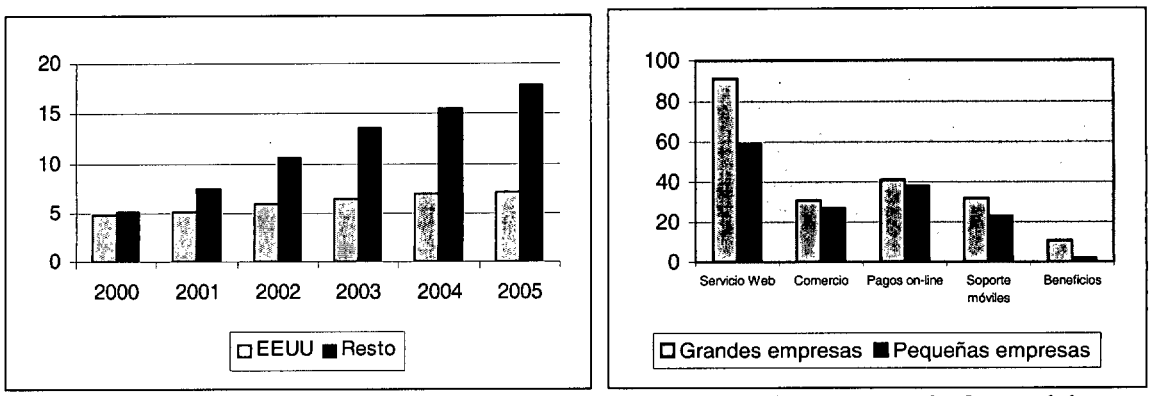


Figura 5. Previsión de crecimiento de sitios Web (millones) y porcentaje de servicios ofrecidos en Web [AUI]

1.4 Futuro del comercio electrónico

1.4.1 Comercio B2B y B2C

Pasada ya la época de euforia y depresión que se vivió durante los años 1999 a 2001 (ver Figura 3), el comercio electrónico está actualmente en una etapa de reflotación donde las inversiones se están haciendo con mucha más cautela que en años anteriores tratando de no caer en los mismos errores que en el pasado.

En la mayoría de los casos, los segmentos del mercado han quedado muy definidos tras el hundimiento de las empresas *punto-com* que surgieron para aprovechar la oportunidad y no supieron adaptarse a la ingente cantidad de pedidos o aparecieron en un segmento que estaba ya muy copado o fueron fácilmente absorbidas por alguno de los gigantes del sector que dominan ahora los segmentos, etc. Hoy en día, parece que el declive de las *punto-com* está tocando fondo (ver por ejemplo [Ros02]) y más que la aparición de nuevas empresas de venta virtual, visto que dicho esquema no es sostenible, lo que se observa es una progresiva tendencia a la adopción de Internet como mecanismo de negocio por parte de empresas existentes que no contaban con participación en este medio.

En cualquier caso, el proceso es lento y aunque ya se ha visto que la adopción del Web por parte de las empresas es algo habitual, no son todavía tan frecuentes las empresas en las que se pueden hacer compras *on-line*. El 85% de las multinacionales declara que sus ventas online no supera el 10% del total de las ventas de la empresa, lo cual sin embargo es muy optimista puesto que en el caso de las PYMES sólo el 24% de las empresas europeas declaran tener páginas Web realmente provechosas, mientras que más de la mitad ni siquiera pueden sacar provecho de sus Webs.

A pesar de todo, las perspectivas del negocio para los próximos años son muy buenas, sobre todo en el sector B2B (43% de las empresas estima que su porcentaje de

ventas online será superior al 70% del total de ventas de la empresa cuando finalice todo el proceso de reconversión al comercio electrónico del que se ha hablado).

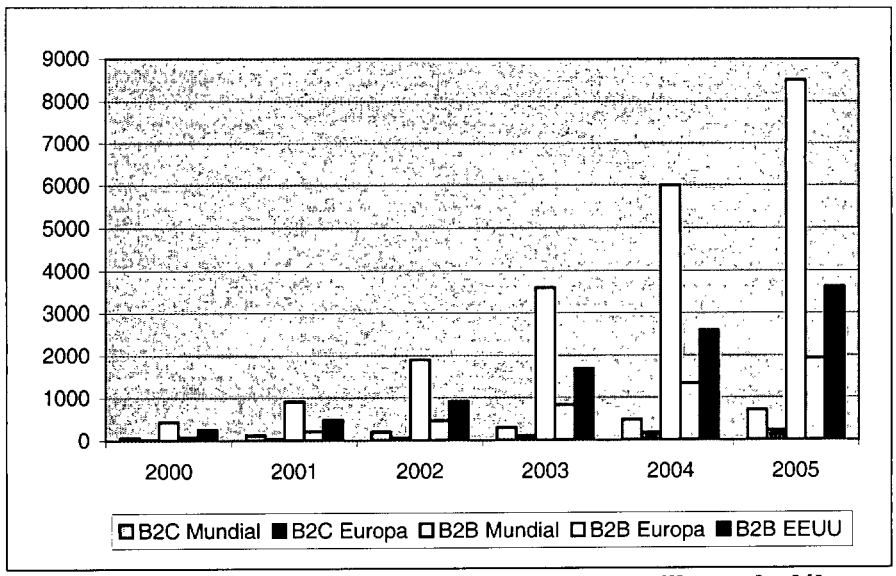


Figura 6. Crecimiento de B2C y B2B en miles de millones de dólares

1.4.2 Comercio móvil

La evolución del comercio móvil, si bien los motivos han sido muy distintos, lo cierto es que ha sufrido unas variaciones semejantes al comercio electrónico típico basado en Internet.

Las plataformas de comercio móvil (*m-commerce*) están actualmente saliendo del período de ralentización al que fueron sometidas tras el desmedido optimismo tecnológico que se planteó con los móviles de tercera generación, la supuesta implantación rapidísima del UMTS, etc.

Sin embargo, al igual que ocurre con el comercio electrónico basado en Internet y habida cuenta de que el número de usuarios de móviles es muy superior al de usuarios de ordenadores personales, las previsiones son muy buenas e indican que el crecimiento del sector en los próximos 5 años irá de los 54 millones de euros de ingresos actuales a 8,6 billones de euros [Garner].

Aunque hoy en día los ingresos del comercio móvil no son muy abultados, la cantidad de usuarios potenciales es tan grande y las previsiones de crecimiento del número de usuarios son también tan grandes que se espera que el comercio móvil empiece a incrementar sus cifras en breve (hoy en día el 45% de los usuarios habituales de Internet declara que ya está utilizando diferentes servicios de comercio móvil).

Hasta el momento, prácticamente todos los ingresos del comercio móvil se deben al servicio de mensajes cortos SMS (*Short Messaging Service*) y que pese a que es una tecnología que tiene ya cerca de 10 años, SMS no se ha empezado a utilizar realmente hasta hace 12-18 meses. El número de mensajes SMS intercambiados en Europa ha crecido de 3 a 15 billones entre 1999 y 2000 y superó los 200 billones de mensajes en 2001 (se estima que la utilización crece a una velocidad del 170% anual).

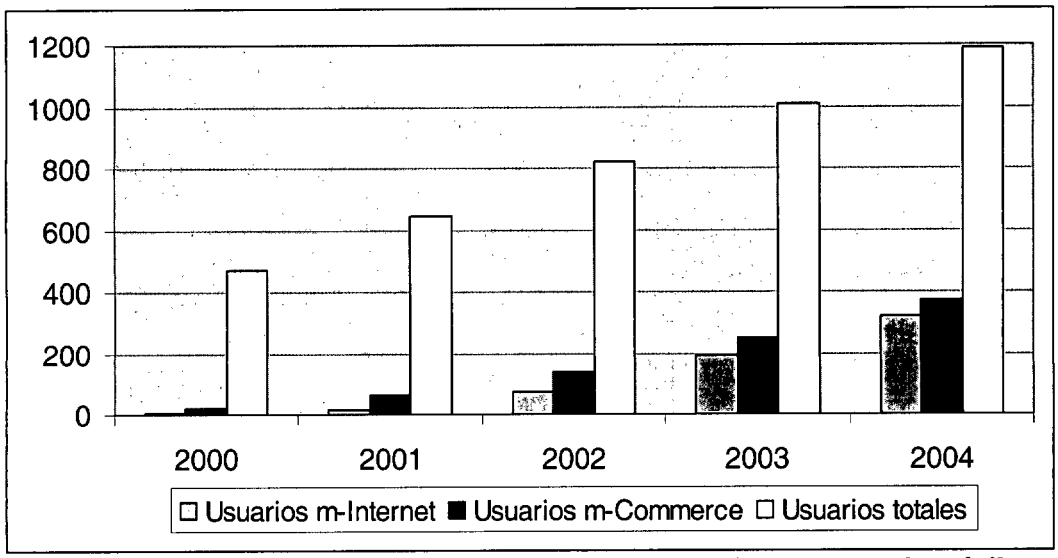


Figura 7. Estimación del número de millones de usuarios de comercio móvil

Aunque al final será sobrepasado por tecnologías como WAP, actualmente es la tecnología dominante en el sector de datos (los cerca de 460 millones de terminales móviles GSM pueden enviar y recibir mensajes SMS) y se espera que lo sea durante cerca de 10 años más. Se estima que en 2004, más del 36% de la población de Europa Occidental utilizará SMS, lo cual supone un 570% de incremento con respecto a 1999 (considerando que por media, un mensaje SMS tiene 45 caracteres y lleva 1kb y cuesta unos 20 céntimos de euro, lo cierto es que el servicio es verdaderamente rentable y mucho más caro que la voz).

	1G	2G		2.5G		3G	
bps	6,4K	GSM 9.6K	14,4K	HSCSD 56K	GPRS 114K	EDGE 384K	UMTS 2M
Voz	+	+	+	+	+	+	+
SMS	-	+	+	+	+	+	+
Mensajería instantánea por mail		-	-	+	+	+	+
Gestión de información personal				-	+	+	+
Acceso Internet, Intranet				-	+	+	+
Anuncios M-commerce				-	+	+	+
Transferencia de ficheros				-	+	+	+
Multimedia pasiva							
Música BD					+	+	+
Vídeo BD					-	+	+
Vídeo en tpo. real						-	+
Multimedia interactiva							
Vídeo conferencia						-	+
Juegos interactivos						-	+
Televisión interactiva							

Tabla 1. Relación entre ancho de banda de las tecnologías y aplicaciones

1.4.3 WAP

Desde su aparición en 1999, el protocolo para aplicaciones inalámbricas WAP (*Wireless Application Protocol*, [WAP]) fue considerado como el 'Internet móvil', pero finalmente debido a diferentes circunstancias como las bajas velocidades de conexión, pobre y escasa calidad de contenidos y el tamaño inadecuado de las pantallas de los terminales, WAP ha resultado ser simplemente un servicio de teletexto, manteniéndose muy por debajo de las expectativas.

En cualquier caso, la realidad es que actualmente hay 50 millones de teléfonos WAP en circulación y al menos 20 tipos diferentes de PDAs con navegadores WAP. Además hay ya más de 8 millones de páginas WAP activas y más de 10000 aplicaciones WAP disponibles.

Para aplicaciones como la *banca móvil*, el protocolo WTLS (*Wireless Telephony Layer Security*), es capaz de ofrecer ya un cifrado de 128 bits, lo cual ha bastado para convencer a los bancos de que la plataforma es segura y ya se ha lanzado también un soporte para infraestructura de clave pública. Para el sector bancario, WAP ofrece tanto la seguridad como la funcionalidad requerida por ofrecer a los clientes gran cantidad de servicios como el pago de facturas con el teléfono móvil, la realización de transferencias, etc. (el Deutsche Bank por ejemplo, está ofreciendo servicios de transferencias, consultas de saldo, obtención de talonarios, posibilidad de acceder al mercado de valores así como diferentes servicios de información sobre vuelos, trenes, motores de búsqueda, etc.).

Se puede concluir, que aunque todavía hay detalles como las prestaciones o la interoperabilidad que hacen que WAP aún no se haya implantado definitivamente y todavía haya dudas sobre su futuro, el hecho de que la tan esperada nueva generación de móviles esté a punto de aparecer, hace que las expectativas con respecto a este protocolo sean muy optimistas (en Marzo de 2002 ha salido la versión 2.0 de WAP, que está basada en xHTML en vez de WML y pretende acercarse más al estándar *i-mode* japonés, con lo que se amplían aún más las posibilidades del protocolo).

1.5 Plataformas de intermediación

1.5.1 Contexto actual

El hecho de que más del 50% de los contenidos accedidos en Internet en España sean páginas de buscadores, directorios, servicios de información, etc., pone de manifiesto que la utilización que actualmente se hace de la red es principalmente como fuente de información y que la cantidad de información es tan grande y su contenido tan heterogéneo, que es imprescindible la utilización de herramientas que faciliten su localización.

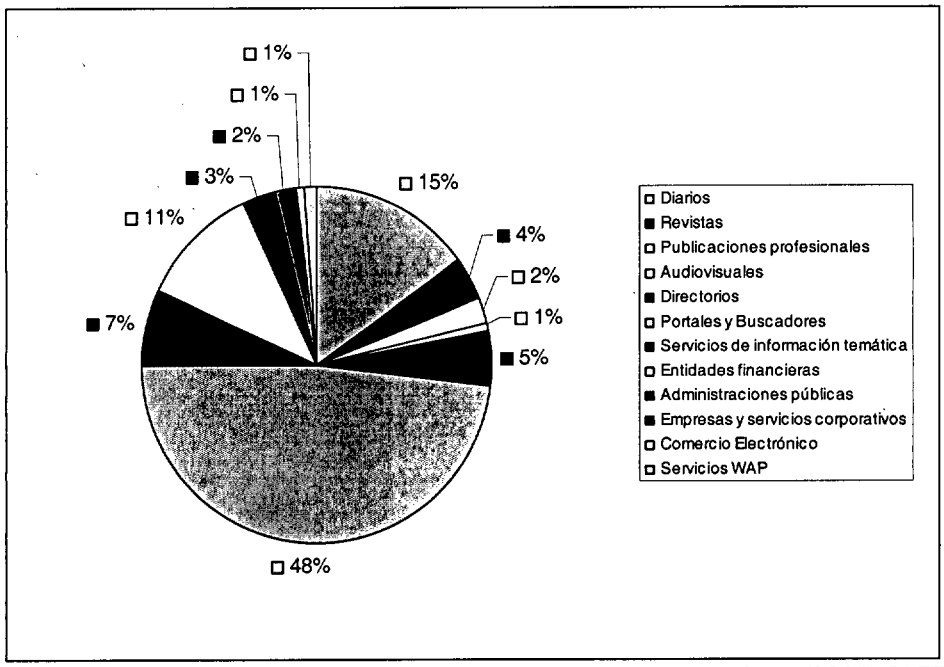


Figura 8. Distribución de páginas Web accedidas por contenidos (2001) [AUI]

En este sentido, son interesantes las conclusiones a las que están llegando diferentes estudios llevados a cabo sobre mecanismos de búsqueda en Internet (ver por ejemplo [Sul02]). Parece razonable que si en un alto porcentaje de ocasiones los usuarios acceden a servicios de búsqueda de información, los accesos al resto de sitios Web provengan también con gran probabilidad de dichos motores de búsqueda.

Sin embargo los estudios vienen a demostrar que el 52% de los usuarios llega a los sitios Web mediante navegación o por introducción directa del enlace en el navegador (típicamente *bookmarks*). Sólo un 7% de los usuarios accede a los sitios Web a través de buscadores.

Este fenómeno (diferentes autores lo denominan '*search gap*'), no le quita en absoluto relevancia a los buscadores. Se puede observar por un lado, el efecto que está teniendo el notable incremento de la publicidad y el marketing de los sitios Web en los medios de información y por otro el hecho de que una vez que los usuarios han encontrado sitios que cubran sus necesidades, pasando generalmente por los correspondientes buscadores, no suelen buscar mucho más y se quedan en él (a partir de ese momento ya saben dónde ir y no necesitan un buscador).

Todas estas observaciones son en general válidas desde el punto de vista del comercio electrónico. La proliferación de aplicaciones y sitios Web de comercio electrónico, la cantidad de productos, de información y de servicios disponible es muy grande y además se va incrementando continuamente (de manera análoga, también es habitual el aumento de la cantidad de información demandada por parte de los clientes como se ha visto).

Todos esos servicios electrónicos están principalmente basados en un intercambio de información eficiente y preciso entre las diferentes entidades implicadas en las transacciones (típicamente cliente y proveedor), usando la infraestructura de comunicaciones existente. Cuando el cliente requiere algo, crea una demanda en el

mercado. Por otro lado, los proveedores proporcionan bienes y servicios, generando ofertas en el mercado.

Es necesario algún tipo de funcionalidad o mecanismo que permita a los clientes y a los servidores intercambiar información de manera fluida y precisa, tratando de minimizar el tiempo que normalmente emplean ambos en encontrar información verdaderamente útil, que suele ser bastante alto.

En el contexto del comercio electrónico, es también imprescindible que una vez que se haya localizado la información, se pueda acceder a diferentes servicios transaccionales adicionales. Una plataforma de intermediación sería precisamente la encargada de llevar a cabo todas estas tareas, actuando como lugar de encuentro entre ofertas y demandas:

- Facilita a los clientes la búsqueda y la puesta en contacto con los proveedores.
- Facilita a los proveedores la búsqueda de clientes y la propagación de sus ofertas.
- Posibilita transacciones comerciales completas con garantías de seguridad (incluyendo pago, etc.).

Esta plataforma de intermediación es por lo tanto una especie de versión electrónica avanzada de unos grandes almacenes (como se verá posteriormente, este tipo de plataformas suele llevar asociados importantes servicios de valor añadido en el campo de la intermediación de información). Los usuarios del servicio serían tanto los clientes (compradores finales) como los proveedores de contenidos (suministradores) y el cometido de la plataforma sería en definitiva, poner a ambos en contacto, hacer búsquedas inteligentes de información (*brokering* o intermediación de información) y facilitar transacciones comerciales seguras como mecanismos de pago o distribución electrónica (ver [Vil+99]).

En esta tesis las plataformas de intermediación se analizan enfocándolas como un modelo de negocio avanzado de comercio electrónico (ver [Tim98] y [Rap02]). Posteriormente (capítulo 6) se comentará más a fondo el tema de los modelos de negocio, ubicando las plataformas de intermediación dentro de los diferentes modelos que pueden definirse y evaluando cómo el modelo de evaluación que se propone en la tesis es también adaptable a otros modelos de negocio.

1.5.2 Implementaciones

Aunque es posible encontrar en Internet una gran variedad de interpretaciones de lo que debe ser un broker (buscadores, portales, intermediadores, servidores de aplicaciones, proveedores de servicios Web, etc.) no es sencillo encontrar una verdadera plataforma con un paradigma de intermediación como el que aquí se perfila.

Dentro del presente marco de especificación, hay varias aproximaciones que pueden utilizarse para implementar un broker de información y muchos de ellos han sido desarrollados en proyectos de investigación financiados por la Comisión Europea a través de diferentes programas como el ACTS [ACTS], el ESPRIT [ESPRIT] (ambos pertenecientes al Cuarto Programa Marco, de 1994 a 1998) o actualmente con programas como el IST [IST] (Quinto Programa Marco, de 1998 a 2002).

La siguiente lista es un ejemplo de algunas de estas aproximaciones tal y como se han implementado en diferentes proyectos:

- Arquitectura de intermediación genérica de referencia con un especial énfasis en las tareas de seguridad, [SEMPER].
- Una federación de motores de búsqueda, con cada buscador especializado en un área de conocimientos determinado buscando información en diferentes proveedores específicos [ABS].
- Intermediarios especializados que ayudan a los clientes a contactar con los proveedores apropiados (sólo buscan a los proveedores y no necesariamente obtienen la información de ellos) [ABROSE] .
- Catálogos homogéneos especializados, donde el valor añadido del broker se encuentra en la capacidad de agregar catálogos procedentes de diferentes proveedores [OSM], [COBRA].
- Buscadores dinámicos inteligentes, donde el broker es capaz de preprocesar y postprocesar las peticiones de los usuarios basados en diferentes criterios. La información puede ser obtenida totalmente de los proveedores pero el intermediario puede dejar esa tarea a los clientes dándoles una lista con los proveedores útiles [ABS].
- Plataforma capaz de interpretar una petición de un cliente, descomponiéndola en los subservicios que considere oportunos y contactando con los proveedores necesarios para resolver todos y cada uno de esos servicios de manera óptima para la petición global [SMARTEC].

1.6 Complejidad en el comercio electrónico

1.6.1 Contexto tecnológico

El auge del comercio electrónico ha hecho evolucionar de forma muy rápida tanto los modelos de negocio como los esquemas de desarrollo de aplicaciones que se utilizaban hasta hace unos pocos años. Eso ha forzado la realización de urgentes revisiones tecnológicas y ha dejado obsoletas a una gran cantidad de arquitecturas y metodologías.

Además de la aparición de numerosas herramientas, enfoques, tecnologías, etc., que en la mayoría de los casos han entrado en mutua competencia, lo que definitivamente ha contribuido a la amalgama tecnológica que actualmente destaca en el sector, ha sido la proliferación de muchos nexos de unión entre diferentes tecnologías.

Inicialmente prácticamente todas las aplicaciones interactivas de Internet (que no estaban basadas en esquemas cliente/servidor propietarios) se establecían sobre sencillos formularios que el cliente podía enviar a un servidor Web. Este procedía posteriormente a pasarle los datos a un programa encargado de analizarlos y de generar algún tipo de respuesta.

Dicho mecanismo es el ya clásico modelo de comunicación basado en CGIs (*Common Gateway Interface*) con un cliente que generalmente era un programa

(navegador o *browser*) capaz de interpretar contenido HTML [HTML], que le era transferido por el servidor de contenidos HTML utilizando el protocolo HTTP [Ber96+]. A la vez este protocolo se utilizaba para devolver los datos al servidor y allí eran procesados por programas que se ejecutaban (o se interpretaban, porque en muchos casos no eran sino *scripts*) en la misma máquina que el servidor.

Posteriormente fueron apareciendo tecnologías que en general iban orientadas a potenciar las funcionalidades del lado del cliente como fue el lenguaje de programación Java [Java], que se utiliza para programar *applets* (pequeñas aplicaciones que se ejecutan por los propios navegadores) o el lenguaje Javascript, que incluido en el propio código HTML es interpretado por el navegador.

Con Java llegó también la posibilidad de utilizar un protocolo alternativo a HTTP para transmitir datos (el protocolo RMI [RMI]). Dicho protocolo inicialmente sólo permitía invocar métodos de forma remota, pero progresivamente fue evolucionando y se le fueron incorporando servicios adicionales (seguridad, servicio de directorios, etc.).

Java se convirtió en el lenguaje estándar de Internet y aunque en muchos casos se ha ido abandonando la utilización de Java en los clientes, por lo pesado que resulta desde el punto de vista de recursos consumidos o del tiempo de ejecución, su presencia en los servidores se ha ido incrementando cada vez más. Primero fueron los *servlets*, que no son sino servidores Web con toda la potencia del procesado Java de soporte y posteriormente los servlets se flexibilizaron con la aparición de la tecnología JSP (*JavaServer Pages*).

Con respecto a la posición de las arquitecturas de distribución, la tecnología se fue moviendo progresivamente hacia esquemas más abiertos, capaces de adaptarse mejor a los requisitos del mercado y de integrarse y reutilizar mejor los diferentes recursos de almacenamiento de datos disponibles. Son las denominadas arquitecturas multicapa.

Arquitecturas como J2EE [J2EE] o .NET [.NET] rivalizan hoy en día por convertirse en la base de la implementación de las futuras aplicaciones de comercio electrónico. Ambas vienen apoyadas por tecnologías bien asentadas como son Java y RMI en el caso de J2EE o DCOM en el caso de .NET y ambas pretenden integrar todas las tecnologías que sus respectivos fabricantes han ido desarrollando a lo largo de estos años y combinarlas con otras tecnologías como XML [XML], que parecen estar implantándose de forma definitiva en el sector. J2EE es la culminación arquitectural del mundo Java desde el punto de vista del soporte a aplicaciones distribuidas de comercio electrónico y .NET es la réplica de Microsoft para adaptar la tecnología del mundo Windows al desarrollo de servicios Web.

Y en todos los casos, siempre se ha tenido un cuidado especial por mantener la 'etiqueta' de la interoperabilidad, haciendo que cualquier arquitectura fuese, al menos en teoría, ampliable y adaptable usando puentes como CORBA [CORBA], que desde hacía tiempo se había consolidado como seguro de interoperabilidad entre mundos tecnológicos heterogéneos.

1.6.2 Iniciativas de comercio electrónico

Además de todos estos detalles de carácter tecnológico, empresas y organizaciones han ido liderando diferentes iniciativas arquitecturales de comercio electrónico con el objetivo de potenciar uno u otro modelo de negocio en función de sus intereses:

- ***Estándares europeos***: en materia de comercio electrónico, los estándares europeos se articulan a través de una rama del CEN creada en 1997, para centrarse en actividades relacionadas con la sociedad de la información: el ISSS (*Information Society Standarization System*, [ISSS]). De entre los proyectos liderados por esta organización, destacan tres: *Building Blocks* [BBlocks], *Architectures and Models for Electronic Commerce* [ECarch] y *E-Commerce Integration Meta-Framework* [ECIMF].
- ***ISOC/IETF***: el IETF (*Internet Engineering Task Force*, [IETF]) es el grupo tecnológico del ISOC (*Internet Society*) que tiene por misión el desarrollo (identificación de problemas técnicos, propuesta de soluciones, etc.) de nuevas especificaciones de estándares para Internet, que se publican en forma de RFC (*Request For Comments*). El trabajo técnico en el IETF está gestionado por el IESG (*Internet Engineering Steering Group*) y dentro de él destacan dos grupos en el área del comercio electrónico: *Internet Open Trading Protocol* [IETFa] (es el grupo del IETF más directamente vinculado con el comercio electrónico y se encarga de desarrollar una especificación de un entorno de interoperabilidad para el comercio electrónico en Internet) y el *Electronic Data Interchange-Internet Integration* [IETFb] (centrado en aspectos relacionados con el intercambio electrónico de datos en Internet de forma genérica, evalúa las diferentes propuestas tradicionales existentes y estudia los requisitos con el enfoque actual de Internet y las posibles sinergias con las tecnologías emergentes, sobre todo con XML).
- ***Unión Europea***: la Unión Europea [UE], a través de la Comisión Europea financia una serie de programas destinados principalmente a fomentar la investigación y el desarrollo en unas determinadas áreas. A estos programas pertenecen los proyectos ya mencionados ABS, ABROSE, COBRA, OSM, SEMPER (programa ACTS) o el proyecto SMART-EC (programa IST).
- ***CommerceNet* [Comm]**: es un consorcio fundado con el objetivo de utilizar Internet como entorno global para la implantación y el desarrollo del comercio electrónico. Hoy en día, es uno de los consorcios más importantes y cuenta con más de 500 miembros en todo el mundo. Los proyectos más destacables de CommerceNet son: *RossetaNet* [Rosetta], *Open Buying in the Internet* [OBI], *E-Commerce architecture* [eCo] y *Universal Marketplaces* [UM].
- ***OMG***: el *Object Management Group* [OMG], es una organización no lucrativa creada en 1989. Actualmente el consorcio incluye más de 800 empresas con el compromiso de crear especificaciones para el desarrollo de software industrial y bajo unos criterios de excelencia técnica, viabilidad comercial e independencia (neutralidad). Para liderar las iniciativas y especificaciones relacionadas con el comercio electrónico, se ha definido un grupo de trabajo concreto: el EC-DTF (*Electronic Commerce Domain Task Force*, [ECDTF]) cuyos resultados más relevantes son: *Negotiation Framework* RFP (*Request for Proposals*), *Public Key Infrastructure* RFP, *Registration and Discovery RFP*.

- ***UN/CEFACT, OASIS***: el UN/CEFACT (*United Nations Centre for Trade Facilitation and Electronic Business*, [UN/CEFACT]) es un comité de las Naciones Unidas establecido en 1996 por la Comisión Económica Europea de las Naciones Unidas, en respuesta a una necesidad de reconocer las contribuciones tecnológicas del sector y de reutilizar los recursos existentes de forma eficiente. Una de sus propuestas más interesantes desde el punto de vista del comercio electrónico, es la iniciativa ebXML, que patrocina junto con OASIS, un consorcio internacional sin ánimo de lucro cuyo principal objetivo es la creación y el fomento de estándares de interoperabilidad para la industria basados en estándares públicos de estructuración de información como XML, SGML y CGM ([OASIS]).
- ***Open Application Group* [OAG]**: es un consorcio no lucrativo formado en 1995 por numerosas empresas líderes en el sector de desarrollo de aplicaciones software. En 1995 desarrolló una arquitectura y una serie de especificaciones englobadas bajo el nombre genérico de *Open Application Group Integration Specification* [OAGIS], que desde entonces ha sido probada y actualizada fomentando su integración con contenidos XML.
- ***Iniciativas empresariales***: por nombrar una de las que más repercusión ha tenido, por tratarse de la plataforma auspiciada por Microsoft Corporation, se puede mencionar Biztalk [Biztalk], cuya arquitectura está especialmente pensada para adoptar una tecnología muy específica, XML, como mecanismo de intercambio de mensajes de negocio, conformes a un formato de datos común sobre una red de comunicaciones como Internet.

1.6.3 Complejidad

Todos y cada uno de los puntos que se han ido tratando, ponen de manifiesto lo *complejo* que resulta hoy en día comprender el comercio electrónico en todas sus facetas (tecnológicamente, económicamente, socialmente, etc.).

Analizando el fenómeno desde la perspectiva tecnológica, el motivo esencial es que el auge del comercio electrónico ha hecho evolucionar de forma muy rápida los esquemas de desarrollo que se utilizaban hace unos pocos años, y eso ha forzado urgentes revisiones tecnológicas y ha dejado obsoletas a una gran cantidad de arquitecturas y metodologías. Además de la aparición de numerosas herramientas, enfoques, tecnologías, etc., que en la mayoría de los casos han entrado en mutua competencia, lo que definitivamente ha contribuido a la amalgama tecnológica que actualmente destaca en el sector, ha sido la proliferación de muchos nexos de unión entre diferentes tecnologías.

El hecho de que haya muchas tecnologías que puedan elegirse para hacer prácticamente lo mismo y la falta de existencia de estándares o tecnologías dominantes en determinadas materias, ha fomentado una diversidad tecnológica a lo largo de estos últimos años que ahora está llevando a la aparición de nuevos mecanismos que permitan la interoperabilidad entre plataformas heterogéneas.

Y todos estos mecanismos no hacen sino sumarse a la ya ingente cantidad de información que las empresas deben absorber de cara a tomar decisiones tecnológicamente acertadas, los desarrolladores deben manejar de cara a crear aplicaciones eficientes y robustas y los educadores deben asimilar para ofrecer una formación actualizada y completa.

Uno de los principales objetivos de la tesis es el contribuir de forma sistemática a analizar toda la complicación tecnológica asociada al comercio electrónico.

Sin embargo, es importante destacar que la tesis no pretende quedarse con el concepto intuitivo de complejidad que se percibe habitualmente y que es el que puede desprenderse de la lectura de los dos apartados anteriores (algo extenso, complicado, cambiante, heterogéneo, difícil de entender, etc.).

La idea es matizar el concepto de complejidad, tratando de definirlo desde un punto de vista teórico y haciendo uso de diferentes modelos creados específicamente para gestionar dicha complejidad [Sae94]. Todo ello permitirá la utilización de diferentes herramientas y artificios teóricos que facilitan el manejo de la complejidad relativa a un determinado elemento estudiado (el comercio electrónico en este caso).

Esto constituye la base del marco de comparación de modelos de comercio electrónico que se definirá en la tesis y cuyo objetivo principal no es otro que el de ayudar a manejar la complejidad asociada al comercio-e.

Dicho marco está orientado a la comprensión, el análisis y la comparación de arquitecturas de comercio electrónico, facilitando la selección tecnológica (bien seleccionar una arquitectura que está siendo objeto de análisis o bien seleccionar una determinada tecnología para una arquitectura que se pretenda desarrollar) y proporcionando un orden pedagógico de presentación de tecnologías que se ha utilizado en la primera parte de esta tesis y que en cualquier caso puede servir de base para la formación de futuros desarrolladores o la para la creación de programas de formación.

1.7 Objetivos principales

El objetivo fundamental de la tesis es el de contribuir a **manejar la complejidad** asociada al mundo de las plataformas de intermediación electrónicas mediante la definición de un modelo de evaluación genérico que pueda ser aplicado en diferentes entornos. En base a ese objetivo principal, se proponen los siguientes objetivos parciales:

- El modelo de análisis se ha dividido en diferentes fases, orientadas cada una de ellas a analizar **diferentes aspectos de la complejidad de las plataformas de intermediación electrónica**. Se propone por tanto un mecanismo capaz de analizar no sólo el carácter tecnológico de dichas plataformas a nivel de tecnologías básicas de soporte o a nivel arquitectural o de implementación, sino también aspectos tan importantes por la complejidad que llevan asociada, como el impacto social, el impacto económico, el impacto medio-ambiental, la complejidad asociada a la nomenclatura y el lenguaje, etc.

- **Disminución de la barrera de entrada por parte de los desarrolladores**: uno de los mayores problemas que presenta el mundo de la intermediación electrónica, entendido bajo la perspectiva del desarrollo de aplicaciones que aborden a bajo nivel la utilización, la integración o la implementación de tecnologías de soporte,

es que presenta una considerable barrera de entrada. El motivo principal, es la gran cantidad de tecnologías que han ido apareciendo a lo largo de los últimos años y la exigencia de que todas ellas sean si no conocidas, sí al menos consideradas cuando se plantea cualquier tipo de proyecto de desarrollo. Además, la literatura sobre el estado del arte de dichas tecnologías, que habitualmente contribuye a rebajar esa barrera de entrada, en este caso suele ser escasa dado lo novedoso de todas ellas. La única fuente de información realmente actualizada es la que pueda encontrarse en Internet. Por todas estas razones, en la tesis se hace un acopio de diferentes tecnologías utilizadas y además de ofrecer referencias sobre muchas ellas, se aportan descripciones de todas ellas con el carácter integrador que hoy en día se aprecia en sus respectivas evoluciones. Si bien la tesis no pretende ser totalmente autocontenida en este sentido, sí se aportan herramientas y se contribuye activamente a que la formación en todas las tecnologías que rodean el mundo de la intermediación electrónica actual se haga con una perspectiva de integración y conocimiento global.

- Se propone también a modo de **aplicación del modelo** de complejidad, la particularización al estudio de una plataforma de intermediación electrónica concreta.

- Desde el punto de vista tecnológico, la tesis profundiza en una de las fases del modelo e incide especialmente en la **complejidad asociada a las comunicaciones** en entornos distribuidos, siempre dentro del marco de las plataformas de intermediación.

1.8 Estructura de la memoria

Para facilitar la lectura de la tesis, se incluye a continuación un breve resumen de cada capítulo.

A lo largo de este capítulo, se ha presentado el ámbito general en el que se enmarca la tesis, repasando de forma objetiva la situación actual del comercio electrónico y la problemática que plantea desde el punto de vista de la motivación y los objetivos que también se han introducido en este **primer capítulo**.

La estructura de los siguientes capítulos de la memoria (dos al seis) responde a la aplicación directa del modelo de análisis que se plantea en el capítulo siete con el objetivo de reducir la barrera de entrada que actualmente suponen las tecnologías que recubren a las plataformas de intermediación.

En el **capítulo dos** se hace un estudio de diferentes tecnologías básicas, que son las que generalmente se manejan en la mayoría de las plataformas de Internet (lenguajes de programación/*script*, lenguajes de descripción de datos, protocolos básicos, mecanismos cliente servidor, etc.).

El **capítulo tres** analiza tecnologías que, con el soporte de las que se presentaron en el capítulo dos, posibilitan la implementación de plataformas distribuidas. Son las

denominadas tecnologías *middleware* que se utilizan actualmente para desarrollar arquitecturas multi-capa y que también se presentan en este capítulo (CORBA, J2EE, .NET, etc.).

El **capítulo cuatro** presenta las diferentes iniciativas que hay actualmente en el campo de las arquitecturas de comercio electrónico, así como las diferentes propuestas que están promoviendo dichas iniciativas.

Para terminar con la introducción técnica, el **capítulo cinco** matiza el concepto de comercio electrónico, introduciendo el concepto de plataforma de intermediación de información como modelo de negocio que ayuda a potenciar la funcionalidad de las plataformas de intermediación electrónica.

El **capítulo seis**, formaliza el concepto de complejidad introduciendo brevemente diferentes teorías y mecanismos de manejo de la misma. En base a los criterios establecidos por determinados modelos de complejidad, se presenta un modelo en cinco fases, para facilitar el análisis de la complejidad asociada a las plataformas de intermediación.

En el **capítulo siete** se ejemplifica la aplicación del modelo mediante el análisis de una plataforma de intermediación electrónica (proyecto SMART-EC), desarrollado dentro del programa de investigación y desarrollo IST, financiado por la Unión Europea.

Por último, el **capítulo ocho** resume las conclusiones principales de la tesis e incluye referencias hacia posibles trabajos futuros, siguiendo la línea del trabajo que se propone en esta memoria.

Capítulo 2

Tecnologías básicas

2.1 Introducción

Este capítulo hace un recorrido por diferentes tecnologías que se utilizan en el entorno de la intermediación electrónica. Desde las soluciones empleadas en los primeros modelos, basados exclusivamente en navegación Web, se irán comentando diversas aportaciones que han aparecido en este campo para construir aplicaciones que contribuyesen a resolver los problemas de acercamiento entre clientes y proveedores (compradores y vendedores).

Todas las tecnologías mencionadas en este capítulo, que se han denominado básicas, exigen ser presentadas, si no descritas en detalle, pues en la mayoría de los casos constituyen el fundamento o son la motivación de otras tecnologías que se verán con posterioridad. En cualquier caso, no es la idea de este capítulo el ofrecer una exposición pormenorizada de cada tecnología, pues la documentación disponible sobre ellas es más que abundante.

*El **objetivo de este capítulo** es el de desarrollar la evolución de cada tecnología así como incluir una breve descripción de cada una de ellas, pero siempre con un carácter integrador que permita iniciar el siguiente capítulo con una visión tecnológica global suficiente como para poder entender la situación actual de lo que hoy en día se ha dado en llamar plataformas middleware.*

Esta presentación de tecnologías se hará en orden de abstracción creciente. Inicialmente se comentarán los lenguajes de programación o de descripción más importantes desde la perspectiva del comercio electrónico. A continuación se hablará de protocolos que habitualmente se utilizan para transferir información (definida en ocasiones con lenguajes de descripción específicos) entre aplicaciones normalmente programadas en los lenguajes anteriormente comentados. Posteriormente se repasan los diferentes tipos de clientes o servidores que se usan para habilitar el funcionamiento de los protocolos anteriores. Por último se detallan los nuevos esquemas de aplicaciones (y servicios) que se ejecutan por encima de los clientes y servidores, y que empiezan a utilizarse actualmente en Internet.

Se pondrá un énfasis especial en tecnologías Java puesto que, según se ha ido viendo a medida que el sector se desarrollaba, hay una clara tendencia a ir adoptando este tipo de soluciones.

En capítulos sucesivos se verá cómo estas tecnologías encajan entre sí para constituir los modelos arquitecturales que hoy en día se empiezan a utilizar de forma generalizada: las arquitecturas multicapa.

2.2 Lenguajes

Desde el punto de vista del desarrollo de aplicaciones de intermediación electrónica, una de las primeras cosas a tener en cuenta son los lenguajes que estarán involucrados en dicho desarrollo. Esto que suele ser así para casi todo tipo de aplicaciones, tiene una relevancia considerable en éste caso debido a las especiales características que rodean al mundo de la intermediación, como son los requisitos de flexibilidad y eficiencia a los que serán sometidos, adaptabilidad, posibilidades de crecimiento y sobre todo la facilidad para interaccionar con módulos arquitecturales diferentes (clientes, bases de datos, otras aplicaciones existentes, etc.).

Pese a que el nombre genérico de 'lenguaje' está abierto a gran cantidad de interpretaciones desde el punto de vista tecnológico, este capítulo se centrará en los lenguajes de programación (más concretamente en Java) y de *scripting* por la relevancia que tienen en el desarrollo de aplicaciones y en los lenguajes de descripción/estructuración de datos (*marcado*), por la proyección y la relevancia que están adquiriendo hoy en día.

2.2.1 Lenguajes de programación

Tal y como se aprecia en la Figura 10, el número de lenguajes de programación de alto nivel que se han desarrollado desde el Fortran de 1954 al C# de finales de 2001 es enorme.

La elección de uno u otro lenguaje está normalmente sujeta al tipo de aplicaciones que se pretendan desarrollar y de hecho hay muchos otros lenguajes que han surgido con

el objetivo claro de potenciar o facilitar cierto tipo de funcionalidades específicas en un determinado entorno.

En el caso de las aplicaciones orientadas a Internet, el lenguaje más popular es sin duda Java. Pese a que inicialmente tardó en erigirse como alternativa seria a C o C++, sobre todo por los problemas de eficiencia que presentaba, el hecho de que esté sustentado sobre una filosofía y una serie de librerías específicamente diseñadas para encajar perfectamente en Internet, hacen que hoy en día sea el lenguaje preferido en este sector.

Además, la línea de evolución que ha seguido Sun Microsystems con Java, una de sus tecnologías estrella y que ha culminado con la aparición de la plataforma J2EE, ha estado siempre orientada a favorecer la integración del lenguaje con Internet.

Actualmente se estima que el número de programadores que utilizan Java superará a lo largo de 2002 a los que utilizan C, C++ o Visual Basic. Más de la mitad de los desarrolladores de Estados Unidos utilizan ya Java, se espera llegar al 60% a finales de 2002 y además, el 78% de las universidades ya enseñan Java (fuera de EEUU las estimaciones son aún más optimistas) [Gal01].

2.2.1.1 Java

En 1990, Sun Microsystems [Sun] inició un proyecto llamado '*Green*' con la idea de desarrollar software para aparatos electrónicos de consumo (VCRs, hornos microondas, sistemas de seguridad, sistemas estéreo, etc.). James Gosling y Patrick Naughton, veteranos diseñadores de software, fueron asociados al proyecto y decidieron empezar a programar en C++. Sin embargo, pronto se dieron cuenta de que C++ se trataba un lenguaje que utilizado bajo la óptica que se planteaba en el proyecto, era muy susceptible a errores que podían bloquear el sistema con relativa facilidad. Concretamente, C++ es capaz de referenciar directamente los recursos del sistema (a base de punteros, etc.) y aunque esto ofrece una potencia y unas posibilidades muy grandes, exige que el programador tenga siempre en cuenta cómo se manejan dichos recursos. Y esto suponía un serio problema a la hora de escribir los programas fiables y transportables que exigía el proyecto, pues debían ser válidos para gran cantidad de aparatos diferentes cada uno con recursos de lo más heterogéneo.

Gosling empezó a trabajar (Agosto de 1991), en el desarrollo de un lenguaje que llamó *Oak* (literalmente "*roble*") y que incorporaba la potencia y la sintaxis de C++, pero evitaba tener que referenciar recursos directamente (se omitía también la aritmética de punteros y la sobrecarga de operadores, que tanto complicaba el trabajo de programadores y depuradores de programas en C y C++). Oak incorporaba además, un gestor de memoria en el propio lenguaje, con lo que se liberaba al programador de esta tediosa tarea. Asimismo, puesto que debía ser útil para muy diversos aparatos, se puso especial hincapié en el apartado de portabilidad.

Poco después (1993-1994), Internet estaba ya en una etapa de tremenda expansión y se pensó que Oak era un lenguaje que encajaba perfectamente con la tecnología y la filosofía existente en la red. Se decidió pues, lanzar el lenguaje por Internet de manera gratuita para conseguir la mayor difusión posible. En Septiembre de 1994 apareció un producto conocido como *WebRunner* que era un navegador programado completamente

en Oak y que pretendía demostrar la potencia de este lenguaje. Finalmente, en Enero de 1995 Oak cambió su nombre por otro más comercial: Java [Java] (y WebRunner se convirtió en el actual *HotJava Browser*). Sun anunció oficialmente Java y HotJava en el congreso SUNWorld'95 (Mayo de 1995).

El lenguaje tuvo tanta popularidad, que cuando Sun lanzó el primer entorno de desarrollo en Enero de 1996, JDK 1.0, Java ya era considerado como el lenguaje de implementación estándar de Internet.

Durante el primer semestre de 1996, un gran número de empresas del sector, entre las que se encontraban Adobe, Borland, IBM, Microsoft, Novell, Oracle, Symantec, etc., solicitaron una licencia de Java. En Mayo de 1996 aparecía la versión 1.0.2 y de ahí se pasó, en Diciembre de 1996, a la nueva versión 1.1 del API. Desde entonces, cada dos meses aproximadamente, fueron apareciendo sucesivas revisiones de este API que incorporaba funcionalidades básicas, así como numerosas APIs adicionales que aumentaban considerablemente la potencia del entorno. A finales de 1999 Sun decide incorporar todas esas funcionalidades dispersas al producto básico y crea un entorno global de desarrollo con la versión 1.2 de sus JDK, que en esta nueva etapa de la plataforma pasa a denominarse "Java 2".

En esta línea de integración se ha continuado desde entonces hasta la versión actual la 1.4 (Figura 9), con la idea de aumentar progresivamente la potencia de la plataforma estándar, en vez de ir segregando librerías independientes como se hacía al principio.

Además, aunque tecnologías como los Servlets o JSPs llevaban ya un tiempo utilizándose (como librerías independientes), desde principios del año 2000 Sun decidió introducir de lleno la tecnología Java en los servidores. La plataforma J2EE (ver [J2EE]), está específicamente diseñada para cubrir los requisitos del lado del servidor en el entorno de desarrollo de las aplicaciones de Internet, ofreciendo por un lado un punto de acceso al cliente (como servidor Web avanzado basado en JSPs) y por otro integrando la lógica de negocio de la aplicación con las correspondientes bases de datos.

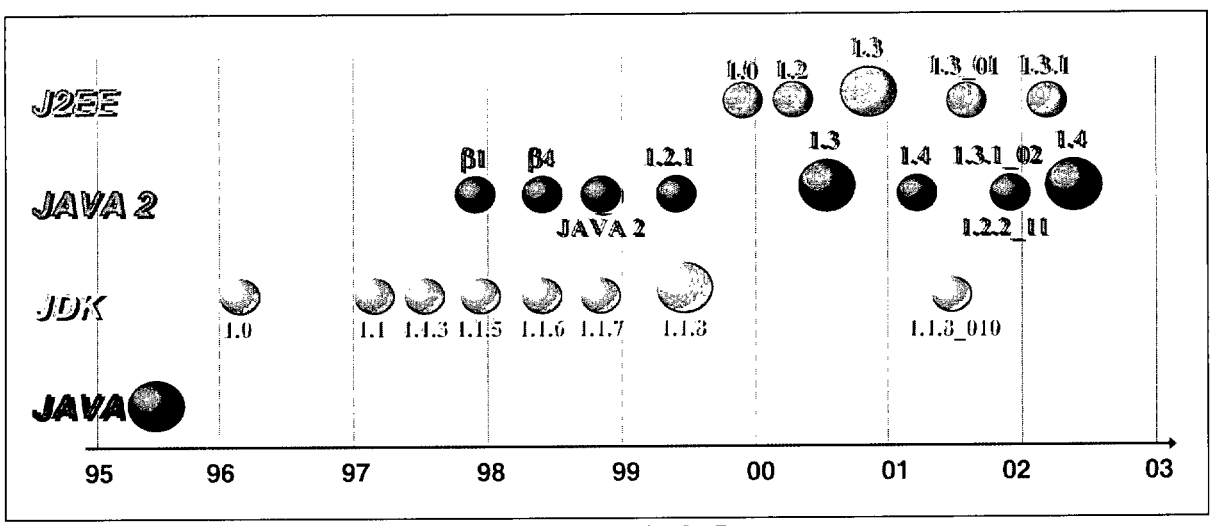


Figura 9. Historia de Java

2.2.1.2 Características

2.2.1.2.1 Portabilidad

Es probablemente la característica más importante de este lenguaje y la que ha determinado la aceptación de Java como lenguaje estándar en Internet. Independientemente del hardware o del sistema operativo sobre el que se esté ejecutando, el desarrollador dispone siempre de las mismas librerías para crear aplicaciones. Además, una vez que se escribe un programa en Java, puede ser ejecutado en cualquier plataforma sin necesidad de cambio o recompilación alguna, al contrario de lo que ocurre con los lenguajes de programación convencionales.

En su plataforma de edición empresarial J2EE, las aplicaciones adquieren una complejidad considerable y suelen tener que interactuar con numerosos módulos, exigiendo por tanto una configuración específica en cada caso. Aun así, la plataforma añade una etapa más al ciclo de desarrollo de software tradicional, que denomina *deployment* (despliegue) y que permite controlar todas estas labores de configurar final (bases de datos, autenticación y autorización de usuarios, propiedades transaccionales, integración con servidores Web o WAP, etc.), sin necesidad de tener que modificar o recompilar el programa Java original.

2.2.1.2.2 Orientación a objetos

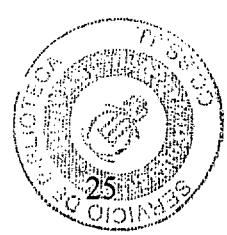
Al igual que el lenguaje que motivó su aparición, C++, Java es un lenguaje de programación de alto nivel orientado a objetos. Si bien simplifica algunas de las características que más definen al lenguaje C++, como puede ser la aritmética de punteros y con ella toda la versatilidad que ofrece en cuanto al manejo de estructuras dinámicas, etc., Java conserva prácticamente toda la potencia de C++. Por otro lado, las simplificaciones introducidas, aumentan la fiabilidad de las aplicaciones, pues entre otras cosas impiden la referencia directa a los recursos de la máquina.

Además, la orientación a objetos (más *fuerte* si cabe que en C++), garantiza una mejor estructuración y posibilita la reutilización de código de una manera sencilla, así como la ampliación y mantenimiento de las aplicaciones que se generen.

2.2.1.2.3 Distribución

En el sistema operativo Windows, determinadas partes de algunos programas, pueden estar representadas por *librerías dinámicas* (DLLs), de modo que pueden ser compartidas por otros programas y cargarse automáticamente cuando sean necesarias. De la misma manera, UNIX permite el uso de *librerías compartidas*. Con ello se consigue, por un lado ahorrar memoria y por otro incrementar la modularidad de las aplicaciones.

Java, amplia este esquema de librerías, permitiendo que las clases necesarias para ejecutar nuestra aplicación, puedan obtenerse de manera dinámica y absolutamente transparente del disco o de cualquier máquina conectada en red.



2.2.1.2.4 Multi-hebra

Esta propiedad, que ya está presente en muchos lenguajes de última generación, adquiere una importancia singular en el caso de Java, pues es un lenguaje en el que la interacción con el usuario juega un papel fundamental y normalmente es necesario mantener activa la interfaz del usuario, mientras se realizan otras tareas.

Desde la perspectiva del servidor, el multiproceso que ofrecen funcionalidades como CGI al tratar los datos que llegan del cliente, no termina de ser una solución completamente escalable y los procesos ligeros de Java son una buena alternativa a este fenómeno.

Por ésta y por otras razones (aplicaciones multiusuario, *timeouts*, etc.), se ha puesto especial hincapié para que la creación de diferentes hebras (*threads*) de ejecución en Java, sea una labor relativamente sencilla de cara al programador.

2.2.1.2.5 Gestión de memoria

Esta es otra de las posibilidades de Java, que viene a solucionar un importante problema que arrastraba la programación en lenguajes como C y Pascal.

Durante la ejecución de un programa, se requiere memoria para poder realizar operaciones temporales como ordenación de listas, tratamiento de imágenes o simplemente para crear los objetos o variables que vayan a utilizarse en algún método. En C y C++, es el programador el encargado de solicitar dicha memoria y de liberarla cuando no la vaya a utilizar más. Si esta tarea no se realiza con precisión, el programa puede terminar su ejecución de inmediato, o ir consumiendo memoria progresivamente hasta agotar por completo los recursos.

Java resuelve el problema con un proceso de baja prioridad (se ejecuta cuando la CPU tiene poca carga y así se aumenta la eficiencia) denominado *recolector de basura* (*Garbage Collector*), que chequea la memoria buscando objetos sin referenciar y los elimina para ahorrar recursos. De esta manera, el programador queda liberado de la tarea de gestionar la memoria.

Lo que sí es importante, es que el recolector de basura no actúa inmediatamente después de que se haya desechado un objeto, sino que se tarda un tiempo en poder reutilizar el espacio de memoria que ocupaba. Es decir, hay que tener cuidado en cualquier caso, de no solicitar grandes cantidades de memoria de manera consecutiva (aunque no más cuidado del que se tendría programando en C o C++).

2.2.1.2.6 Prestaciones

Una de las características más criticadas de Java es la velocidad de ejecución que puede ofrecer en comparación con lenguajes como C o C++. El motivo principal es que mientras que C y C++ son lenguajes que pasan por una compilación que traduce y adecua el código fuente para ejecutarse de manera óptima en la plataforma elegida, en Java el proceso de compilación lo que genera es un código intermedio que luego es interpretado por la JVM.

Pese a que en los inicios de Java lo cierto es que esas diferencias de prestaciones eran muy notables, precisamente por ser una de las cosas más criticadas, ha sido también una de las cosas que más se han mejorado a medida que se iban desarrollando nuevas versiones de máquinas virtuales. Progresivamente se han ido introduciendo mejoras en la arquitectura, como el JIT (*Just in Time Compiler*) o el motor HotSpot (ver [HotSpot]) que han aumentado considerablemente las prestaciones de toda la plataforma Java acercándose mucho a las de C o C++ y ahora al nuevo lenguaje C# [C#] y [Oba01] (Tabla 2).

Versión de JRE/JDK	Grado de lentitud con respecto a C++
v1.0	20-40
v1.1	10-20
v1.2	1-15
v1.3	0,7-4
v1.4	0,5-3 (típicamente 1,2-1,5)

Tabla 2. Comparación de la velocidad de Java y C++ [Mar02]

La siguiente tabla, muestra a modo de resumen algunas características del lenguaje comparándolas con otras alternativas:

	Java	Smalltalk	Tcl	Perl	Shell	C	C++
Simple	●	●	●	◐	◐	◐	●
OO	●	●	●	●	●	●	◐
Robustez	●	●	●	●	●	●	●
Seguro	●	◐	◐	●	◐	●	●
Interpretado	●	●	●	●	●	●	●
Dinámico	●	●	●	●	◐	●	●
Portable	●	◐	●	●	◐	◐	◐
Neutral	●	◐	◐	●	◐	●	●
Threads	●	●	●	●	●	●	●
Recolecc. Basura	●	●	●	●	●	●	●
Excepciones	●	●	●	●	●	●	◐
Prestaciones	●	◐	●	◐	●	●	●

Tabla 3. Comparación de Java con otros lenguajes de programación [GMc96]

2.2.1.3 Arquitectura Java

El núcleo de la arquitectura Java lo constituye la llamada máquina virtual Java (JVM), que es un ordenador abstracto capaz de ejecutar los ficheros *.class* generados por el compilador Java, que contienen código binario denominado *bytecode*, de muy bajo nivel (en vez de obtener el clásico fichero ejecutable, en Java se obtienen estos ficheros .class). Estos bytecodes son de hecho las instrucciones en código máquina de la JVM que es en definitiva, la encargada de ejecutar las aplicaciones Java.

Internamente, los bytecodes se ejecutan gracias a una serie de registros, una pila de operadores, un área reservada para cada uno de los métodos del programa y un montón de recolección de basura donde se crean los objetos durante la ejecución del programa.

El término 'virtual', se deriva de que normalmente esta máquina es simulada por software sobre plataformas existentes (Windows, Linux, Unix, etc.). Sin embargo, existen también implementaciones hardware de máquinas Java (como la *picoJava* [Sun02b]), que permiten aumentar la eficiencia de la ejecución.

La gran novedad que ofrece este esquema, consiste en que los mencionados bytecodes, son independientes de la arquitectura hardware, pues la JVM se ejecuta sobre dicho hardware ocultándolo. Esta es precisamente, la base que sustenta la tremenda portabilidad comentada en el apartado de características.

Lo que evidentemente sí es dependiente de la arquitectura es la JVM (que normalmente se viene programando en C), aunque Java está ya tan sumamente extendido, que hay implementadas máquinas virtuales prácticamente para cualquier sistema operativo (incluidos los que se ejecutan en algunos móviles, PDAs, etc.).

2.2.1.4 Integración Web

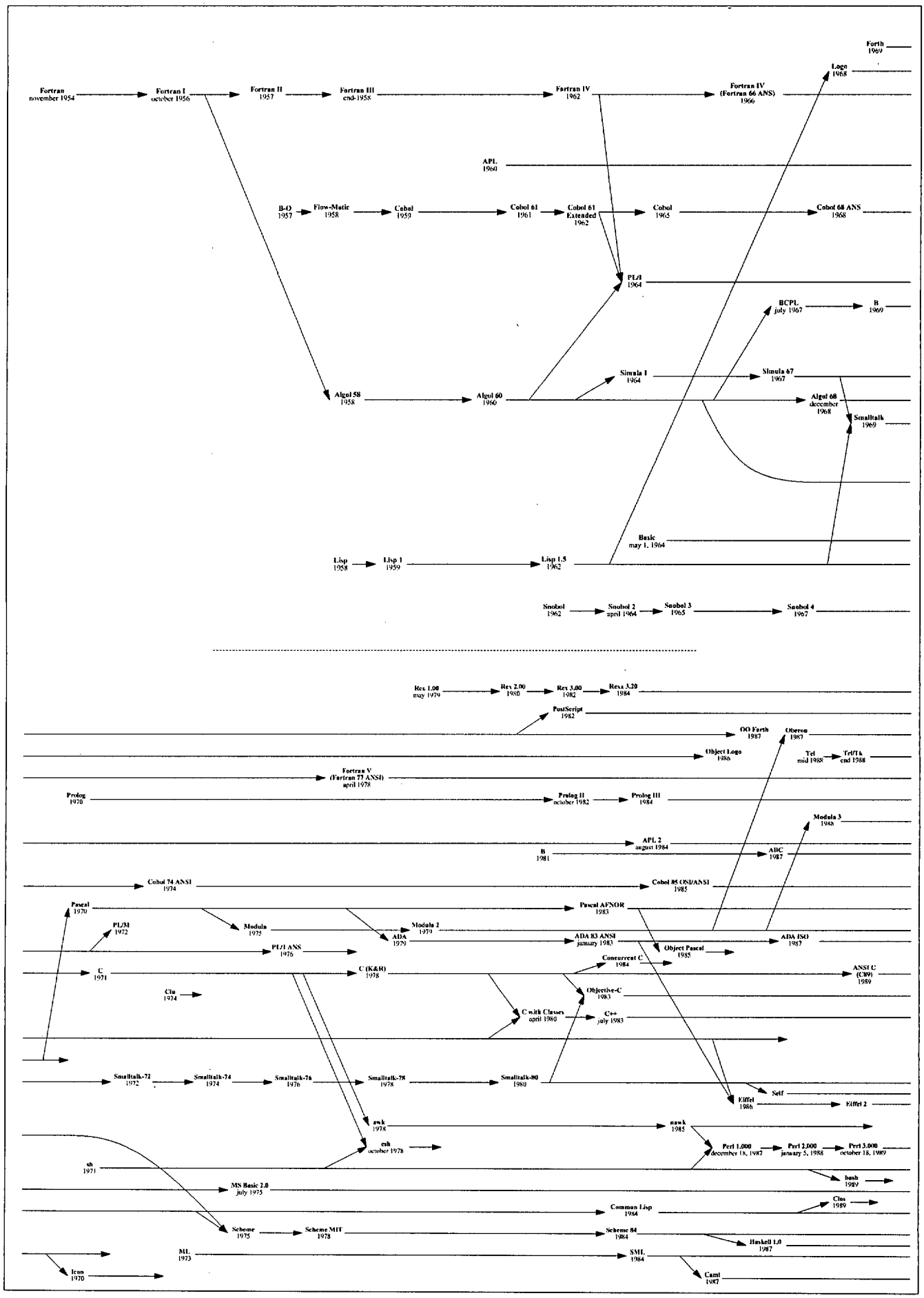
De cara a integrar perfectamente Java con Internet, desde el principio los navegadores comerciales más conocidos (Netscape Navigator e Internet Explorer), implementaron su propia JVM adaptándola a la plataforma en la que se fuese a utilizar.

Netscape lanzó el primer navegador (1.0) en octubre de 1994 y un año después, su versión 2.0, ya soportaba Java y JavaScript. Ya desde el principio el navegador de Netscape se comprometió a respetar el estándar de facto que marcaba Sun, a diferencia del IExplorer, que en un intento de disputarle a Sun la primacía en el sector de Java, hizo algunas variaciones en el lenguaje que fueron motivo de actuaciones legales y de la supresión del logotipo de compatibilidad con Java. Todo esto hizo que prácticamente todas las aplicaciones Java del momento fuesen desarrolladas y utilizadas bajo el entorno compatible del navegador de Netscape.

Inicialmente y hasta su versión final 4.0 (febrero del 97), dicho navegador sólo soportaba el API 1.0.2 de Java. De la versión 4.01 a la 4.05 y pese a que en las últimas ya se anunciaba compatibilidad con el API de la versión 1.1 de Java, era necesario la utilización de unas clases adicionales que ofrecía Sun para ejecutar applets que usasen ésta interfaz. Las últimas versiones del navegador tienen incluido el soporte Java hasta la versión 1.1.5 de las JDK y ya es posible ejecutar applets basados en esta versión de Java sin necesidad de clases añadidas.

Sin embargo, dada la complicación que le generaba a Netscape el hecho de tener que ir implementando su propia máquina virtual e ir actualizándola a medida que Sun publicaba nuevas versiones, a raíz de la aparición de Java 2, decidieron abandonar el proceso de actualización de la máquina virtual y desde entonces solamente mantienen la de la citada versión 1.1.5.

Actualmente lo más habitual para la ejecución de applets en un navegador, es la utilización de un *plugin* que proporciona Sun con cada nueva versión de Java que lanza al mercado y que es capaz de integrarse satisfactoria tanto con el Navigator como con el IExplorer (recientemente Sun ha vuelto a potenciar la tecnología de los clientes mediante la tecnología Java WebStart™ [Sun02e], pero lo cierto es que aún no está muy difundida).


Fortran
november 1954
Fortran I
october 1956
Fortran II
1957
Fortran III
end-1958
Fortran IV
1962
Fortran IV
(Fortran 66 ANS)
1966
Forth
1969
Logo
1968
APL
1960
B-O
1957
Flow-Matic
1958
Cobol
1959
Cobol 61
1961
Cobol 61
Extended
1962
Cobol
1965
Cobol 68 ANS
1968
PL/I
1964
BCPL
july 1967
B
1969
Simula I
1964
Simula 67
1967
Algol 58
1958
Algol 60
1960
Algol 68
december
1968
Smalltalk
1969
Basic
may 1, 1964
Lisp
1958
Lisp 1
1959
Lisp 1.5
1962
Snobol
1962
Snobol 2
april 1964
Snobol 3
1965
Snobol 4
1967
Rex 1.00
may 1979
Rex 2.00
1980
Rex 3.00
1982
Rexx 3.20
1984
PostScript
1982
OO Forth
1987
Oberon
1987
Object Logo
1988
Tcl
mid 1988
Tcl/Tk
end 1988
Fortran V
(Fortran 77 ANSI)
april 1978
Prolog
1970
Prolog II
october 1982
Prolog III
1984
Modula 3
1988
APL 2
august 1984
B
1981
ABC
1987
Cobol 74 ANSI
1974
Cobol 85 OSI/ANSI
1985
Pascal
1970
Pascal AFNOR
1983
PL/M
1972
Modula
1975
Module 2
1979
ADA
1979
ADA 83 ANSI
january 1983
ADA ISO
1987
PL/I ANS
1976
Object Pascal
1985
Concurrent C
1984
C
1971
C (K&R)
1978
ANSI C
(C89)
1989
Clu
1974
Objective-C
1983
C with Classes
april 1980
C++
july 1983
Smalltalk-72
1972
Smalltalk-74
1974
Smalltalk-76
1976
Smalltalk-78
1978
Smalltalk-80
1980
Self
Eiffel
1986
Eiffel 2
awk
1978
nawk
1985
csh
october 1978
Perl 1.000
december 18, 1987
Perl 2.000
january 5, 1988
Perl 3.000
october 18, 1989
sh
1971
bash
1989
MS Basic 2.0
july 1975
Common Lisp
1984
Clos
1989
Scheme
1975
Scheme MIT
1978
Scheme 84
1984
Haskell 1.0
1987
ML
1973
SML
1984
Icon
1970
Caml
1987

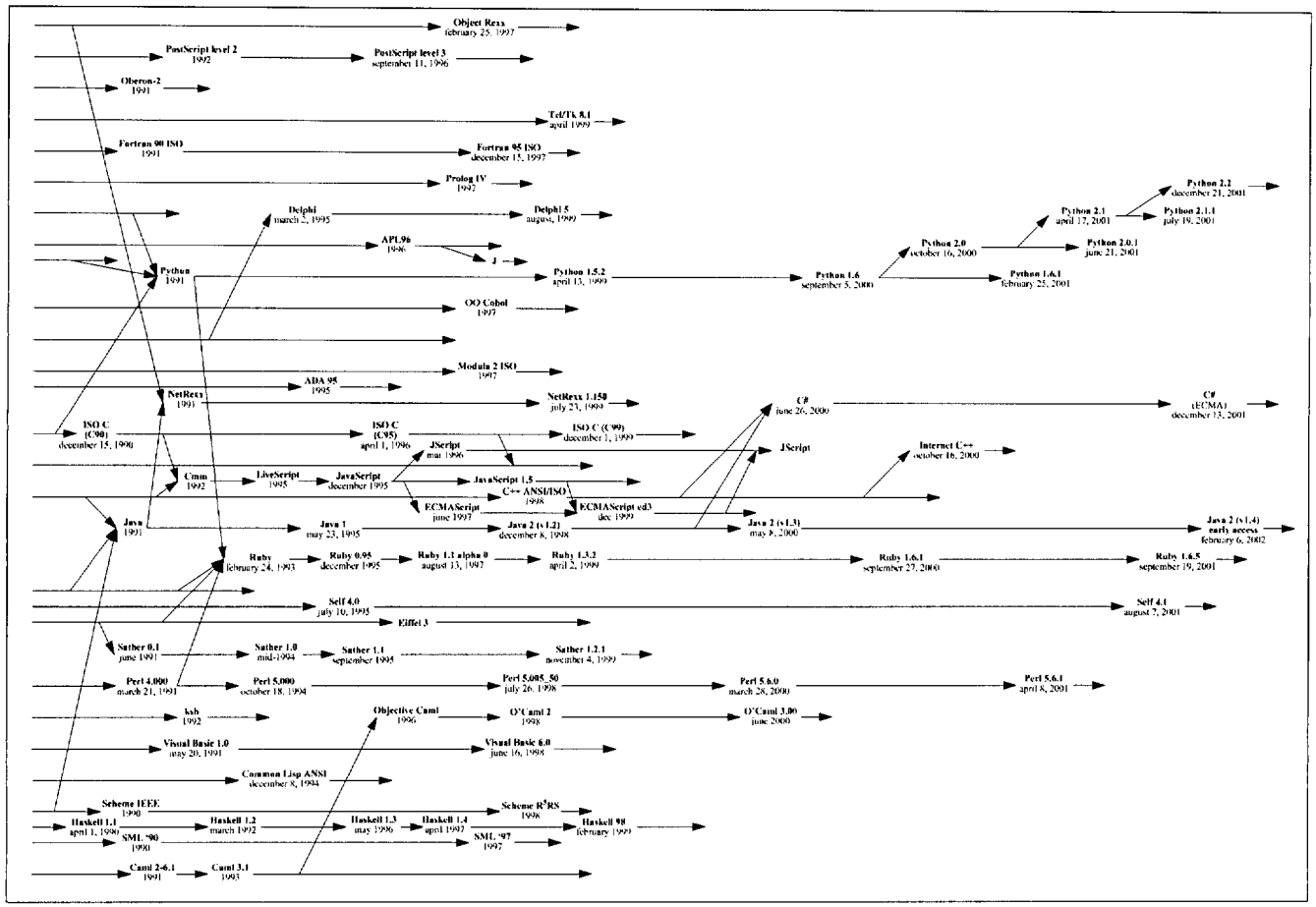


Figura 10. Historia de los lenguajes de programación y de *scripting* [Lev02]

Con respecto a la complejidad de los detalles técnicos, el proceso para descargar una aplicación Java (que al ejecutarse en un navegador pasa a llamarse *applet*) de un servidor Web (ver posteriormente en la Figura 13), es similar al de la transferencia de cualquier otro tipo de fichero que vaya integrado en una página HTML (ver 2.2.3.1), por lo que no supone ninguna complejidad adicional a los protocolos de transporte existentes, utilizados por los navegadores.

Una vez descargado el applet, la máquina virtual incorporada en el navegador es capaz de llevar a cabo las acciones oportunas para ejecutar el programa Java.

2.2.2 Lenguajes de scripting

Tan pronto como surgen los primeros lenguajes de programación de alto nivel (lenguajes que son capaces realizan ciertas tareas de las denominadas de 'bajo nivel' de forma automática, es decir, que cada una de sus instrucciones equivalen a 5-10 instrucciones ensamblador), empiezan a surgir también lenguajes, inicialmente muy vinculados a los sistemas operativos, para facilitar la integración de componentes y de aplicaciones (el lenguaje JCL, probablemente el primer lenguaje de script, surge en los años 60 y a principios de los 70 empiezan ya a aparecer las primeras shells de Unix).

La principal diferencia con los lenguajes de programación, es que habitualmente estos lenguajes de script siempre parten de la existencia de una serie de aplicaciones construidas con los lenguajes de programación, siendo su objetivo el de mejorar el

acoplamiento e integración de éstas aplicaciones y no el de generar aplicaciones por sí mismos.

Otra diferencia importante, es que los lenguajes de script, generalmente no se compilan, sino que se interpretan, con las ventajas de flexibilidad y desventajas de rendimiento que esto conlleva.

En general los lenguajes de script suelen considerarse de mayor nivel que los lenguajes de programación de sistemas (su equivalente en instrucciones ensamblador es mayor) y además de cara a favorecer las tareas de integración, las exigencias con respecto al tratamiento de los tipos de datos son menores (Figura 11).

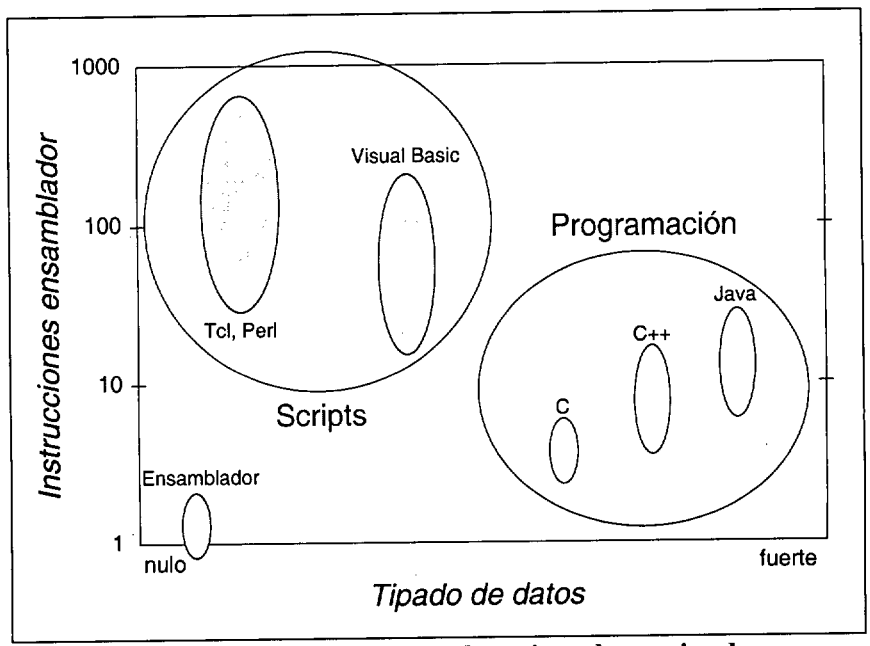


Figura 11. Comparación entre lenguajes de script y lenguajes de programación

De entre los lenguajes de script con más popularidad se pueden destacar:

- ***Shells de Unix***: la primera de ellas (sh) se escribió a principio en 1971 (ver Figura 10) y a partir de ella han evolucionado otros lenguajes de script y han aparecido nuevas shells como csh o ksh. Estas shells (suelen traducirse como 'intérpretes de comandos') se utilizan tanto para escribir comandos interactivos como para escribir scripts que automaticen diferentes tareas. Uno de los principales motivos por el que Unix (y posteriormente Linux), es una de las plataformas preferidas por los desarrolladores de aplicaciones, es precisamente porque estas shells ofrecen la posibilidad de crear aplicaciones nuevas a partir de aplicaciones existentes de forma sencilla.
- ***Perl***: este lenguaje surgió en 1987 con la idea de aunar las funcionalidades que ofrecían diferentes intérpretes de comandos de Unix como sh, sed o awk y pronto adquirió gran popularidad entre los administradores de sistemas. Dicha popularidad se hizo aún mayor cuando con la llegada del Web se convirtió en uno de los lenguajes preferidos para escribir scripts CGI (ver 2.4.3.2).
- ***Tcl***: se creó a finales de los 80 como un lenguaje de comandos fácilmente empotrable en aplicaciones interactivas. Sin embargo este lenguaje alcanza su verdadera difusión cuando se une con Tk y se convierte en el mecanismo más

rápido para crear interfaces gráficas en entornos Unix (a mediados de los 90 se transportó Tcl-Tk a Windows y a Macintosh).

- ***Visual Basic***: el lenguaje de programación BASIC (Beginner's All Purpose Symbolic Instruction Code) surgió a mediados de los años 60 y se extendió cuando Microsoft empezó a incluirlo en las diferentes versiones que sacaba de sus sistema operativo MS-DOS. En línea con la idea del lenguaje interpretado y no compilado que se tenía con BASIC, VisualBasic es un lenguaje que se queda entre la generación de scripts y los lenguajes de programación y que se difundió tremendamente a principio de los 90 en combinación con VBX (posteriormente ActiveX), pues permitía generar interfaces gráficas para Windows muy fácilmente.
- ***Python***: originalmente se diseñó como un lenguaje de script que permitiría salvar las distancias de integración existentes entre el lenguaje C y la programación basada en interpretación de comandos (shell). Sin embargo, pronto se empezaron a aprovechar también sus posibilidades en el desarrollo rápido de aplicaciones Web y aplicaciones de bases de datos con componentes gráficas.

```
<HTML>
  <HEAD>
    <SCRIPT language="JavaScript">
      <!--
      function chequeo(theForm) {
        if (theForm.elements[0].value == "" || theForm.elements[1].value == ""){
          alert("Por favor, introduzca los datos que faltan")
        } else {
          if(theForm.elements[2].value < 1 || theForm.elements[2].value > 120) {
            alert("Esa edad no está permitida")
          } else {
            theForm.submit()
          }
        }
      }
      -->
    </SCRIPT>
    <TITLE>Ejemplo de formulario chequeado con Javascript</TITLE>
  </HEAD>
  <BODY>
    <FORM>
      <H3>Por favor, introduzca su nombre:
      <INPUT TYPE="text" NAME="nombre" SIZE="20"><BR>
      y su apellido: <INPUT TYPE="text" NAME="apellido" SIZE="20"><BR>
      y finalmente, su edad : <INPUT TYPE="text" NAME="edad" SIZE="3"><br>
      <INPUT TYPE="button" VALUE="Enviar" onClick="chequeo(this.form)"></H3>
    </FORM>
  </BODY>
</HTML>
```

Figura 12. Ejemplo de *script* Javascript para tratar un formulario en el cliente (parte superior) y *CGI* en Perl para tratar el formulario en el servidor (parte inferior)

```
#!/usr/local/bin/perl
use CGI;
$q = new CGI;
$nombre = $q->param('nombre');
$apellido = $q->param('apellido');
print "Content-type:text/html\n\n";
print <<END_OF_MESSAGE;
<HTML>
  <HEAD>
    <TITLE>Ejemplo de recepción de página</TITLE>
  </HEAD>
  <BODY>
    <H3>Su subscripción ha sido recibida</H3>
      Muchas gracias, $nombre $apellido, por rellenar el formulario
  </BODY>
</HTML>
END_OF_MESSAGE
```

- ***Javascript***: así como Perl se convirtió en el lenguaje preferido del Web para generar CGIs, Javascript se convirtió también rápidamente en el estándar de facto como lenguaje de script en el lado del cliente. Pese a que al principio no contaba con el apoyo de Microsoft (fue lanzado por Netscape con cierto soporte por parte de Sun), el hecho de que resultaba muy fácil de aprender y de integrar en entornos Web (se integra dentro del propio código HTML que se transfiere de los servidores) hizo que pronto fuese utilizado de forma masiva. El lenguaje permite potenciar las funcionalidades del clásico cliente Web y además es posible la interacción con applets de Java. En el ejemplo de la Figura 12, se muestra un *script* que permite chequear un formulario antes de ser enviado y detectar errores que antes se tenían que detectar en el servidor lo cual propiciaba perdidas de tiempo y de ancho de banda. A partir de este lenguaje, Microsoft implementa *JScript*, que aunque similar en funcionalidades, no es totalmente compatible con *Javascript*.

2.2.3 Lenguajes de marcado de datos

Los lenguajes de marcado de datos surgen en los años 80 cuando con SGML, *Standard Generalized Mark-up Language* [SGML], se pretende facilitar el intercambio de documentación técnica y otras formas de datos publicables entre autores y editoriales. Para ello se plantea la utilización de un lenguaje neutral capaz de proporcionar una definición formal de todos los componentes integrantes de un conjunto de información publicable.

Tan pronto como se desarrolla la tecnología Web se observa que la filosofía de SGML se adecuaría perfectamente a este entorno y surge el lenguaje HTML como un conjunto muy sencillo de elementos para representar información basada en el paradigma del hipertexto.

Sin embargo, a medida que el Web evolucionaba, se vio la necesidad de sistematizar el mecanismo de intercambio de información estructurada y se pensó en XML como el lenguaje que permitiría el intercambio de documentos SGML genéricos.

El lenguaje adquirió tal popularidad que además haberse convertido en el estándar de facto para representar información distribuida en el Web, ha sido aceptado como método general de intercambio y estructuración de información entre ordenadores conectados a Internet. Es considerado como la mejor solución para el intercambio de metadatos sobre objetos y programas almacenados y además se está utilizando también como base para el intercambio de información comercial (definición de procesos de negocio, etc.).

El hecho de que el lenguaje HTML se haya redefinido ya por completo utilizando XML (XHTML), que los grandes fabricantes de navegadores Web hayan declarado que sus productos soportarán XML y que grandes empresas como Microsoft, IBM, Sun o Netscape estén distribuyendo de forma gratuita herramientas de desarrollo para XML, lleva a pensar que la implantación de este protocolo de forma definitiva tanto a nivel profesional como a nivel de usuario, es ya un hecho.

2.2.3.1 HTML

2.2.3.1.1 Características

Cuando en 1991 el CERN lanzó el Web, pronto se vio la necesidad de utilizar algún lenguaje de descripción y estructuración de datos del estilo de SGML, pero que se adecuase a los sencillos terminales que se usaban entonces para acceder a la red (tanto a nivel gráfico como funcional). En 1992 apareció la primera versión de HTML (*HyperText Markup Language*) como un subconjunto de SGML y rápidamente se convirtió en el estándar de facto para la representación de hipertexto en el World Wide Web.

Como lenguaje de marcado que es, los documentos se construyen a partir de etiquetas (*tags*), que son textos cortos que se escriben entre los símbolos '< >' y que caracterizan el texto que va entre una etiqueta de apertura y su correspondiente etiqueta de cierre. Dichas etiquetas le indicarán al intérprete de HTML cómo presentar el texto que está entre ellas. Por ejemplo, el texto `<EM>texto resaltado</EM>`, sería presentado por el navegador en negrita y en itálica. De alguna manera, se trata de abstraerse de la presentación visual del documento y concentrarse en el diseño lógico de las partes del mismo.

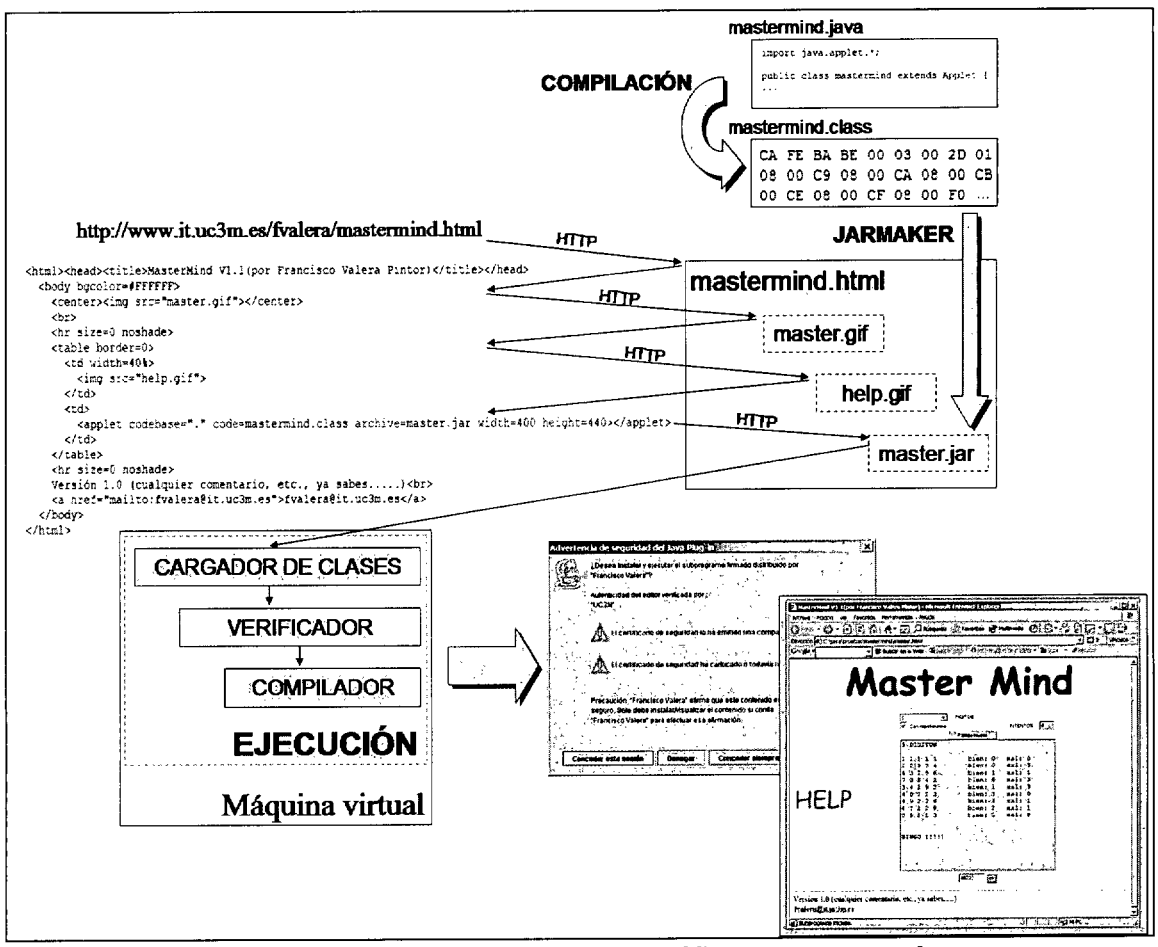


Figura 13. Documento HTML con gráficos y con un applet

Por otro lado, los documentos HTML están constituidos por caracteres ASCII, exactamente igual que si fuese el código fuente de cualquier lenguaje de programación, lo cual facilita su desarrollo y su transmisión. Es decir, un documento HTML no lleva incorporados ni gráficos, ni sonido, ni animaciones, ni applets de Java, ni cualquier otra cosa que no sea exclusivamente texto plano.

Lo que sí lleva para referirse a este tipo de ficheros, son las correspondientes etiquetas que explícitamente hacen que el navegador le pida, mediante el protocolo HTTP, esos gráficos y documentos adicionales al servidor WWW (ver Figura 13). Una vez se han transferido al cliente, bien el mismo navegador, o bien alguna aplicación auxiliar lanzada por él (plugin), se encargará de presentar los archivos de la manera adecuada.

De cara a interaccionar con el usuario, para obtener datos de él por ejemplo, HTML ofrece la posibilidad de crear formularios (FORMs) que pueden ser rellenaos por el usuario y posteriormente enviados al servidor utilizando el método POST del protocolo HTTP. Una vez en el servidor, puede ser tratados con algún programa CGI o con un servlet, como se verá a continuación (2.4.3.2).

2.2.3.1.2 Versiones de HTML

Después de la primera versión de 1992, apareció la versión HTML+ en 1993 (como Internet Draft) aunque no fue hasta 1995 con HTML 2.0 (RFC1866) que el IETF estandarizó el lenguaje y recogió las diferentes prácticas llevadas a cabo hasta el momento por los diferentes fabricantes.

Posteriormente aparecen diferentes actualizaciones (HTML 3.0 en 1994, etc.) y en 1996 el W3C lanza la recomendación HTML 3.2 en cooperación con los más conocidos fabricantes del momento (Microsoft, Netscape y Sun entre ellos).

Esta versión, además de tratar de normalizar la utilización de unas determinadas etiquetas que estaban fuera del estándar, pero que eran de uso habitual, introducía también la idea del DTD, *Document Type Definition*, en HTML (actualmente es parte integral de XML, ver 2.2.3.2).

Finalmente aparece la versión 4.0 como recomendación en 1997, que evoluciona hasta la última versión de HTML que apareció como tal, en Diciembre de 1999: la versión 4.01 [HTML].

En la actualidad se está produciendo una fusión entre el mundo HTML y el mundo XML, de tal forma que la siguiente gran versión de HTML (XHTML 1.0), que no es otra cosa que una reformulación de HTML 4.0 como una aplicación de XML de tal forma que es más fácilmente interpretable y además es totalmente adaptable a los navegadores actuales siguiente unas sencillas reglas.

En mayo de 2001 aparece lo que hasta el momento puede considerarse la última recomendación al respecto: XHTML 1.1 [XHTML] (reformula XHTML de una forma estricta obligando a eliminar cualquier información sobre el formato del texto del documento XHTML y relegando esta funcionalidad a la utilización de páginas de estilos CSS, ver 2.2.3.2.3).

2.2.3.2 XML

2.2.3.2.1 Introducción

XML (*eXtensible Markup Language*) [XML] es, al igual que HTML, un lenguaje de *mark-up* que se utiliza para definir los denominados 'documentos XML'. Dichos documentos se construyen basándose en *entidades* que no son sino unidades de datos encerradas entre etiquetas que les proporcionan estructura. Los procesadores de XML son los programas encargados de proporcionar acceso a la estructura del documento y a su correspondiente contenido y suministrárselo a la aplicación que los necesite.

Este lenguaje fue creado en 1996 por un grupo de trabajo del W3C liderado por Sun Microsystems y con la contribución del grupo de trabajo que anteriormente se dedicaba al desarrollo de SGML. Los principales objetivos de XML son:

- Ser directamente utilizable en Internet.
- Ser fácilmente interpretable por programas.
- Dejar cerrada la especificación en el sentido de que los parámetros opcionales deben de ser mínimos.
- Ser compatible con SGML.
- Ser fácilmente legible por humanos.
- Ser de un diseño formal y conciso.

XML ha recibido tal aceptación, que no sólo se ha convertido en el estándar de intercambio de datos de Internet, sino que está siendo adoptado como tecnología base de gran cantidad de estándares actuales (servicios Web, protocolos de comunicación distribuida como SOAP, servicios de directorios, descripción de procesos de negocio, etc.).

2.2.3.2.2 Documentos XML

Son documentos de texto que siempre empiezan con un patrón de datos denominado *prólogo* que declara al documento XML como tal (`<?xml version="1.0"?>`). Dicho prólogo además de incluir la versión de XML a la que el documento se ajusta, también puede incluir otros parámetros como el juego de caracteres a utilizar o una indicación sobre si el documento es auto-contenido desde el punto de vista de definición de tipos.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Proveedores SYSTEM
    "proveedores.dtd">
<Proveedores>
  <Proveedor>
    <Referencia>10023</Referencia>
    <Nombre>Viajes Águila</Nombre>
    <Direccion>Gran Vía 84</Direccion>
    <CodigoPostal>28012</CodigoPostal>
    <Ciudad>Madrid</Ciudad>
    <Oferta Type="number">12</Oferta>
    <Oferta Type="number">15</Oferta>
    <Oferta Type="number">19</Oferta>
    <Oferta Type="number">40</Oferta>
  </Proveedor>
</Proveedores>
```

```
<!DOCTYPE Proveedores [
  <!ELEMENT Proveedores (Proveedor)>
  <!ELEMENT Proveedor (Referencia,Nombre,
    Direccion,CodigoPostal,Ciudad,Oferta+)>
  <!ELEMENT Referencia (#PCDATA)>
  <!ELEMENT Nombre (#PCDATA)>
  <!ELEMENT Direccion (#PCDATA)>
  <!ELEMENT CodigoPostal (#PCDATA)>
  <!ELEMENT CIUDAD (#PCDATA)>
  <!ELEMENT Oferta (#PCDATA)>
  <!ATTLIST Oferta Type (number)#REQUIRED>
]>
```

Tabla 4. Ejemplo de documento XML y DTD para definir proveedores

Después del prólogo, el documento incluye una sección para la definición genérica de los elementos de marcado que se van a utilizar. Dicho marcado en XML tiene por objetivo la descripción de la estructura lógica del documento, la descripción semántica del texto encerrado dentro de las etiquetas y la asociación de pares atributo-valor con la estructura lógica.

Para la mencionada definición de elementos de marcado, la definición de restricciones en la estructura lógica del documento o para usar entidades predefinidas, XML proporciona un mecanismo específico: la declaración de tipos (DTD, *Document Type Declaration*), que debe aparecer antes de cualquier entidad en el documento (`<!DOCTYPE ...>`).

Dicha declaración de tipos permite asegurar que un documento XML se construya de forma válida definiendo la gramática que se va a utilizar (etiquetas que pueden ser incluidas, atributos posibles para esas etiquetas, restricciones para el contenido del texto entre etiquetas, etc.). El documento XML puede o bien tener una referencia externa a un fichero que incluye el DTD (caso del ejemplo de la Tabla 4) o bien incorporar toda la declaración dentro del propio documento XML.

El resto del documento XML está formado por entidades o elementos que vienen definidos por las etiquetas de inicio y terminación que se han declarado en el DTD. El contenido del documento debe estar incluido dentro de una entidad principal, denominada entidad raíz del documento (*document root*).

Las etiquetas en XML vienen a definir la estructura jerárquica y sintáctica del documento y en vez de definir *cómo* se van a *representar* los datos que encierran (tal y como se hace en HMTL, por ejemplo), definen *qué son* los datos que encierran (por ejemplo, en vez de indicar que el texto va en negrita, con `<b>texto</b>` indica qué es el texto, `<edad>texto</edad>`). El usuario es libre de utilizar las etiquetas que mejor se adecuen a su aplicación (las etiquetas las *inventa* el usuario), aunque evidentemente, para que un documento XML pueda ser utilizado por diferentes aplicaciones, es necesario que se utilice un convenio de etiquetas común (cada convenio es lo que se denomina aplicación de XML y suelen estar definidas en forma de DTD por consorcios y organizaciones industriales).

A diferencia de lo que ocurre en HTML en donde no era necesario que a una etiqueta de apertura le correspondiese un terminador (Ej.: etiqueta `<p>`), e incluso era posible cerrar una etiqueta con el terminador de otra etiqueta distinta, en XML cada vez que se abre una etiqueta, hay que cerrarla posteriormente para garantizar la correcta estructuración del documento.

Esta imposición, junto con la obligación de que las etiquetas estén *completamente* anidadas (si se abre una etiqueta dentro del cuerpo de otra, hay que cerrarla antes de que se cierre la de nivel superior), hace que los documentos XML tengan una estructura jerárquica muy marcada lo cual permite que sean fácil y rápidamente accesibles y modificables.

Una excepción a la regla anterior es la posibilidad de utilizar etiquetas vacías para marcar puntos en el documento, aunque dichas etiquetas tienen un formato diferente a las anteriores (Ej.: `<punto/>`).

2.2.3.2.3 Representación de la información

Las hojas de estilos permiten definir cómo representar (en pantalla, en una impresora, etc.) los datos contenidos en un documento XML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
   <xsl:template match="/"> <xsl:apply-templates/> </xsl:template>
   <xsl:template match="text()|@*"> <xsl:value-of select="."/> </xsl:template>
   <xsl:template match="Proveedores">
      <html>
         <head><title>Texto de salida del formato XSL</title></head>
         <body>
            <h1>Lista de proveedores</h1>
            <ul>
               <xsl:for-each select="./Proveedor" order-by="+ Nombre">
                  <li><xsl:value-of select="Nombre"/></li><br/>
               </xsl:for-each>
            </ul>
            <hr/>
            <h2>Detalles de los proveedores</h2>
            <xsl:apply-templates/>
         </body>
      </html>
   </xsl:template>
   <xsl:template match="Proveedor"><br/>
      <b><a><xsl:attribute name="nombre">#<xsl:value-of select="./Referencia"/>
         </xsl:attribute>
      </a></b>
   </xsl:template>
   <xsl:apply-templates/> <hr width="25%" align="left"/> </xsl:template>
   <xsl:template match="Nombre"> <xsl:apply-templates/><br/> </xsl:template>
   <xsl:template match="Direccion">  <xsl:apply-templates/><br/> </xsl:template>
   <xsl:template match="CodigoPostal">  <xsl:apply-templates/><br/> </xsl:template>
   <xsl:template match="Ciudad"> <xsl:apply-templates/><br/> </xsl:template>
   <xsl:template match="Oferta"> <xsl:apply-templates/><br/> </xsl:template>
</xsl:stylesheet>
```

Tabla 5. Fichero XSL que genera código HTML

Básicamente hay dos mecanismos definidos por el W3C para especificar los estilos de un documento: CSS (*Cascade Style Sheets,* [CSS]) y XSL (*eXtensible Style Language,* [XSL]). Ambos son utilizables en documentos XML, pero mientras que el primero existe desde hace tiempo (1996) y es aplicable incluso a documentos HTML, el segundo que se basa en el anterior y en DSSSL (*Document Style Semantics and Specification Language*), está orientado a documentos altamente estructurados que incluso pueden necesitar ser traducidos a otros formatos antes de presentarlos (desde simplemente resaltar texto, a ponerlo en formato PDF, traducirlo a HTML o WML, etc.).

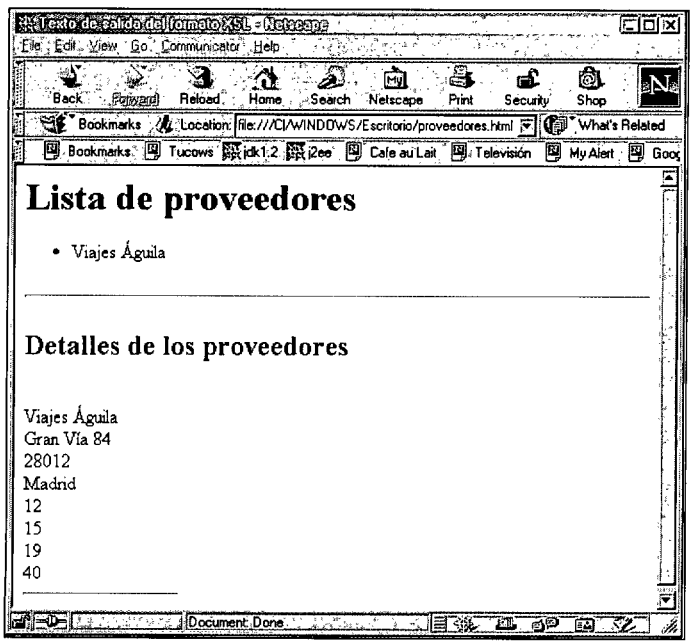


Figura 14. Aspecto final del documento XML tras aplicarle la transformación XSL

2.2.3.2.4 Familia de especificaciones XML

El W3C tiene intención de ir completando la definición de XML mediante la publicación de sucesivas recomendaciones. Hasta el momento se han definido las siguientes:

- ***XML***: define como declarar y utilizar los componentes de un documento XML
- ***XML Namespaces***: permite que elementos que han sido definidos en DTDs diferentes puedan mezclarse sin problema de que haya colisión con los nombres, gracias a identificadores únicos que se asignan a cada DTD y que se incluyen en cada URI a la hora de referenciar un elemento.
- ***XLink (XML Linking Language)***: es la especificación que indica la forma de definir enlaces dentro del marco XML.
- ***XPointer (XML Pointer Language) y XPath***: especificaciones que permiten que la localización de información dentro de un documento referenciado por un enlace pueda hacerse de manera mucho más precisa.
- ***XSL***: además de definir una semántica para especificar cómo se quiere que una determinada clase de documentos sea formateada (para su posterior visualización), se define también un lenguaje para transformar documentos XML (XSLT, *XSL Transformations*).
- ***XML Schema***: pretende jugar el papel de los DTDs de definición de los datos que un documento XML puede contener (valida un documento XML) pero añadiendo a la definición una jerarquía de la que los DTDs carecen, lo cual permitiría una definición mucho más precisa de los datos.
- ***XML Fragment Interchange***: define un formato que permite la transferencia entre ordenadores de partes (fragmentos) de documentos XML.

2.2.3.2.5 XML y Java

Una de las características que tienen en común XML y Java, es la portabilidad: así como Java ya se ha visto que proporciona portabilidad en el código que se envía, XML proporciona portabilidad en los datos que se envían.

Las APIs que define Sun para manejar XML (ver [Sun01]), permiten escribir aplicaciones Web en Java y obtener los beneficios de ambas tecnologías. Tanto es así, que Sun ha sacado un paquete específico para desarrollar servicios Web que integra entre otras cosas, todas las APIs de tratamiento de documentos XML que se habían definido hasta el momento (ver [Sun02d]).

Dichas APIs se utilizan o bien para manejar documentos XML (JASP o JAXB) o bien para utilizar protocolos de comunicación basados en XML (JAXM, JAXR, JAX-RPC).

Los siguientes puntos definen sucintamente en qué consisten estas APIs:

- ***JAXP*** (*Java API for XML Processing*): esta API facilita el tratamiento de documentos XML desde aplicaciones Java, dando cobertura a los mecanismos de análisis estándar como son SAX (*Simple API for XML*) y DOM (*Document Object Model*). Así, se puede decidir entre traducir el documento XML a una cadena de

eventos o bien a un objeto directamente utilizable en Java. La última versión de JAXP es capaz de gestionar también XSLT proporcionando por tanto control sobre la presentación de los datos (y haciendo posible su transformación por ejemplo a HTML o WML).

- ***JAXB*** (*Java API for XML Binding*): permite construir clases Java a partir de un documento XML y su correspondiente DTD (en el futuro se ampliará el API para dar soporte a los esquemas XML).
- ***JAXM*** (*Java API for XML Messaging*): envía mensajes SOAP por Internet desde la plataforma Java con los mecanismos convencionales de dicho protocolo en su versión 1.1 (ver 3.9.5).
- ***JAXR*** (*Java API for XML Registries*): posibilita el acceso a registros estándares de negocio basados en XML a través de Internet como registros ebXML (4.4) o registros UDDI.
- ***JAX-RPC*** (*Java API for XML-based RPC*): permite realizar y recibir invocaciones remotas a través de Internet utilizando SOAP, es decir escribir aplicaciones Java que utilicen XML para realizar RPCs.

2.3 Protocolos

2.3.1 Introducción

En 1974, en un esfuerzo por normalizar las comunicaciones en las redes, las organizaciones de estandarización ISO (*International Standard Organization*) y ANSI (*American Nacional Standards Institute*) crearon un modelo de red dividido en siete capas: el modelo OSI (*Open System Interconnect*).

Desde 1983, año en que aparece Internet, surge también como modelo de soporte a las tecnologías de transmisión de datos, la pila de protocolos TCP/IP. Dicha pila tiene una filosofía modular del estilo de la que ofrece la torre OSI pero está muy orientada a la arquitectura de tecnologías que tiene Internet.

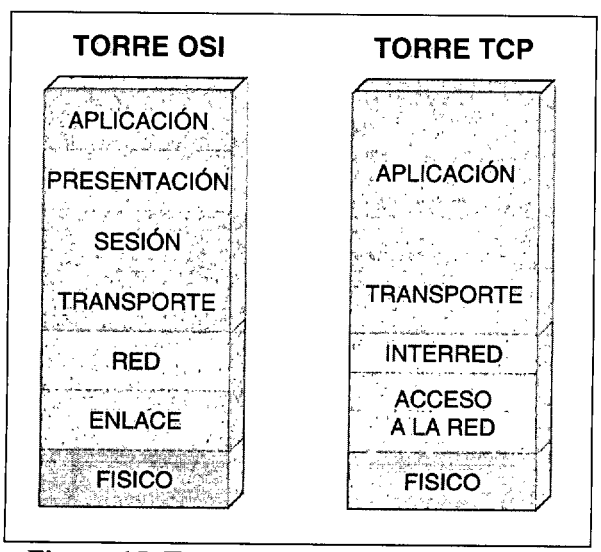


Figura 15. Torres de protocolos OSI y TCP

La idea de la arquitectura de capas o niveles, que debe ser implementada en mayor o menor medida por todos los nodos de la red, es que cada nivel es capaz de proporcionar servicios específicos al nivel superior basándose en los servicios proporcionados por el nivel inferior (ver [Tan96]).

Cada capa puede comunicarse con la inmediatamente superior o inferior por medio de las interfaces definidas al efecto y además pueden comunicarse con capas del mismo nivel ubicadas en nodos diferentes mediante lenguajes específicos, que son los que se denominan ***protocolos***.

La torre de protocolos que se ha acabado imponiendo desde el punto de vista práctico es la torre TCP/IP y hoy en día casi todos los sistemas operativos de utilización masiva incorporan estos mecanismos para dar soporte a la comunicación de datos.

Pese a que la lista de protocolos de nivel de aplicación construidos sobre las facilidades que proporcionan los niveles de red y transporte es interminable, desde el punto de vista de su utilización en el entorno de intermediación electrónica que aquí se plantea, el número de protocolos a considerar está mucho más acotado (HTTP, JRMP, IIOP, SOAP, etc.).

Por la estrecha vinculación existente entre protocolos como JRMP o IIOP con tecnologías de distribución de objetos que se verán más adelante (son protocolos definidos específicamente para sustentar dichas tecnologías), se ha preferido dejar su explicación para apartados posteriores.

Se comenta en este apartado HTTP, no sólo ya por ser el protocolo por excelencia utilizado en el Web o no tener una vinculación específica con ninguna tecnología en particular, sino que además es un protocolo que sirve de base para otros muchos protocolos y mecanismos que se han definido (CGI) o que están en proceso de definición y aceptación (SOAP).

2.3.2 HTTP (HyperText Transmision Protocol)

2.3.2.1 Introducción

HTTP (*Protocolo de Transmisión de Hipertexto*) [HTML], es un protocolo de nivel de aplicación para sistemas de información hipermedia distribuidos, empleado en Internet desde 1990, que constituye la base del Web y que posibilitó la comunicación entre clientes y servidores cuando el comercio electrónico empezaba a desarrollarse. En la actualidad además, es el mecanismo que está sirviendo de soporte para otros muchos protocolos que han ido surgiendo desde entonces, algunos de los cuales se comentarán en apartados sucesivos (protocolos de nivel de aplicación se *montan* sobre HTTP utilizándolo como protocolo de transporte fundamentalmente por su sencillez y por la posibilidad de pasar a través de *firewalls*). Precisamente por este motivo entraremos más a fondo a ver los detalles del protocolo.

La versión inicial del protocolo (0.9) era extremadamente sencilla y únicamente estaba orientada a la transmisión de un flujo de datos a través de Internet. Gradualmente

se fue mejorando el protocolo hasta que apareció la versión 1.0 [Ber96+], normalizando de alguna manera lo que a lo largo de los años se había considerado un estándar de facto y recogiendo las prácticas más comunes que bajo el nombre de HTTP 1.0, venían implementándose de manera diferente en diversos servidores Web. Esta nueva versión, permitía ya que el cliente pidiese una página o la cabecera de la misma al servidor (directivas GET y HEAD respectivamente) o enviase datos a una URL (directiva POST). Y todo ello utilizando el formato MIME, que permitía enriquecer los mensajes con meta-información sobre los mismos. Sin embargo, según se ha ido viendo a lo largo de los años, esta versión del protocolo tenía gran cantidad de problemas que requerían una rápida solución (ver 2.3.2.3).

Desde Enero de 1997 y posteriormente en una actualización de Junio de 1999 [Fie+99] se especifica una nueva versión de HTTP, la 1.1. Esta nueva versión, además de mejorar ostensiblemente el rendimiento de la especificación anterior, incluye un mayor número de métodos que hacen que el protocolo sea mucho más versátil.

En los siguientes apartados se comentará tanto el funcionamiento del protocolo como algunos detalles sobre la implementación del mismo.

2.3.2.2 Funcionamiento del protocolo

2.3.2.2.1 Introducción

HTTP es un protocolo basado en el paradigma petición/respuesta (son los dos únicos tipos de mensajes definidos en el protocolo). El cliente realiza una petición al servidor compuesta principalmente de un identificador de petición, una URI (*Universal Resource Identifier*) y la versión del protocolo, seguido todo ello del cuerpo del mensaje en formato MIME, que puede contener modificadores de petición e información del cliente. La respuesta del servidor es una línea de estatus junto con la versión del protocolo seguida de un mensaje tipo MIME con información del servidor y por fin el contenido de los datos (cuerpo del mensaje). Dicha respuesta es interpretada por el correspondiente navegador de Internet y puede así ser representada en la pantalla del usuario.

En la Figura 13 se puede ver el mecanismo completo desde que el cliente introduce la dirección en su navegador hasta que ve interpretada la respuesta.

Habitualmente el protocolo funciona sobre TCP en el puerto 80 (aunque ambas cosas pueden variarse siempre y cuando se respete el requisito de HTTP de ser transportado sobre un protocolo fiable).

Los próximos apartados ofrecen de forma esquemática las principales estructuras que permiten la construcción de mensajes HTTP, que se ilustrarán finalmente con un ejemplo.

2.3.2.2.2 Peticiones HTTP

El formato general de las peticiones HTTP (ver Figura 16), consta principalmente de un método que será convenientemente interpretado por el servidor, el identificador de un recurso (URI) al que va asociado el método y una serie de cabeceras que permiten matizar el funcionamiento del método o dar información adicional al servidor.

Métodos:

- ***Options***: permite pedir información sobre las posibilidades de comunicación disponibles. La respuesta (estatus 200) incluirá las cabeceras necesarias para indicar qué opciones están disponibles en el servidor para un recurso determinado.
- ***Get***: permite obtener el recurso identificado por el URI. Es posible hacer un 'Get' condicionado (poniendo las cabeceras 'if' correspondientes) e incluso varios 'get' parciales en los que, especificando el rango correspondiente (range), no es necesario coger el cuerpo entero del servidor (estas condiciones son útiles para implementar mecanismos de caché, por ejemplo).
- ***Head***: obtiene por respuesta las cabeceras que se generarían de haber hecho un 'get', pero no obtiene nada del cuerpo del mensaje.
- ***Post***: se le pide al servidor original que acepte los datos que se le envían en la petición para que sean tratados por el recurso especificado en el URI (para una BD, el envío de datos correspondientes a un formulario a un CGI, etc.).
- ***Put***: se le pide al servidor que almacene los datos que se le envían en el URI correspondiente (ya sea creación o modificación del recurso).
- ***Delete***: permite borrar el recurso especificado por el URI (el cliente no tiene garantías de que se haya ejecutado con éxito, incluso aunque el código de estatus así lo indique).
- ***Trace***: le permite al cliente determinar qué se recibe exactamente al otro lado y utilizarlo como información de diagnóstico. La respuesta debe contener en su cuerpo la petición entera y como 'Content-type', message/http.
- ***Connect***: permite modificar opciones de la conexión (un proxy puede ser utilizado como un túnel, etc.).

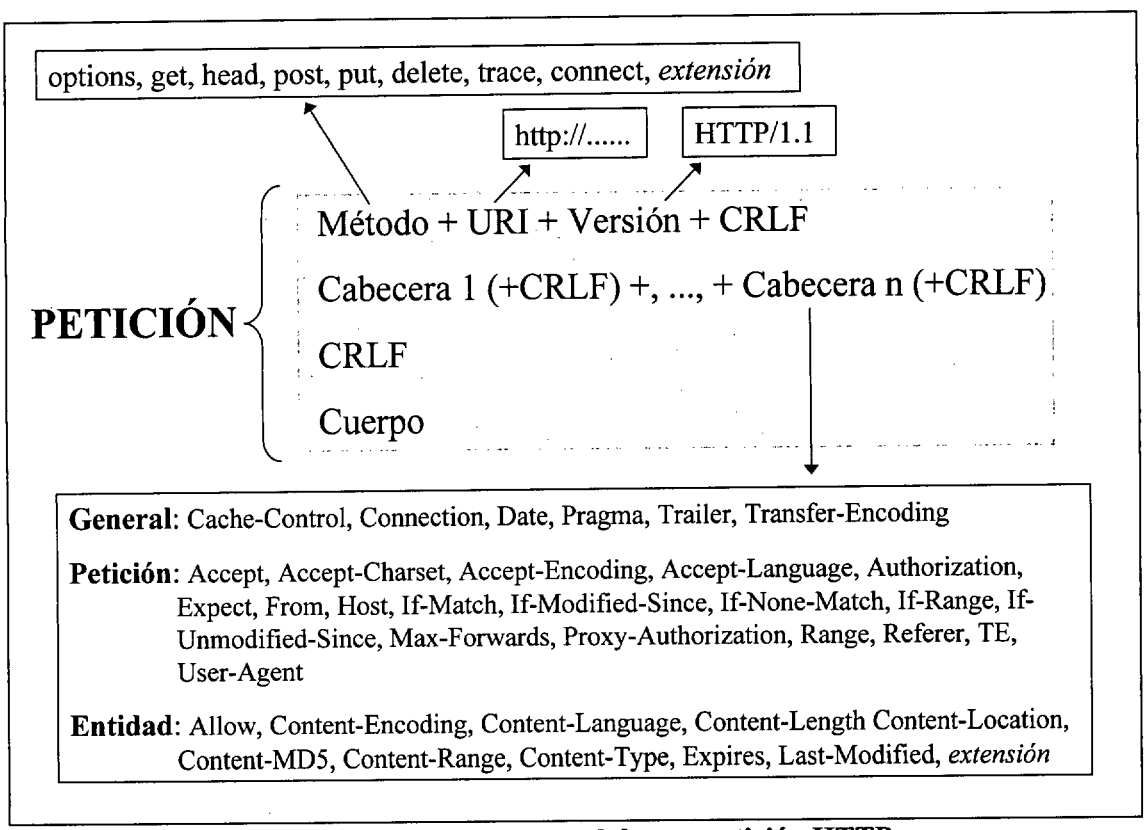


Figura 16. Formato general de una petición HTTP

2.3.2.2.3 Respuestas HTTP

Las respuestas HTTP (ver Figura 17), generadas siempre como consecuencia de una petición, incluyen inicialmente un código y un mensaje de estatus que le permite al cliente conocer si la petición se realizó con éxito o hubo algún tipo de incidencia durante su procesamiento.

2.3.2.2.4 Cabeceras

Las cerca de cincuenta cabeceras, permiten añadir información a las peticiones o respuestas sobre el contenido del mensaje, sobre diferentes opciones o sobre modificaciones disponibles a la comunicación en curso, etc.

Dichas cabeceras pueden ser específicas de la petición o la respuesta, o bien genéricas de cualquier mensaje HTTP.

2.3.2.2.5 Autenticación

El mecanismo de autenticación de HTTP/1.0 (*`Basic Authentication`*) es comúnmente considerado como poco seguro (a no ser que se use junto con SSL u otro mecanismo externo), pues tanto el identificador de usuario como la contraseña que se utiliza, se envían sin cifrar por la red. La nueva especificación [Fra+99] define el '*`Digest access authentication`*' que también se basa en una técnica de secreto compartido (en ningún caso se define cómo llegar a compartir dicho secreto), pero que no transmite claves en claro.

Autenticación básica: después de recibir una petición sobre un recurso dentro de un área protegida, el servidor responde con un mensaje de estatus '`401 Unauthorized`' y adjunta la correspondiente cabecera de autenticación para instar al cliente a que introduzca sus datos de seguridad ('`WWW-Authenticate: Basic`' '`realm=Nombre_del_Realm`'). Para ser autorizado, el cliente debe enviar como credencial el login y el password, válidos en el entorno del servidor al que pertenezca el recurso referenciado en la petición, separados por '`:`' y codificados en base64 (ej: *`Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==`*).

Autenticación basada en resumen: igual que en el caso anterior, cuando el cliente pide un recurso protegido, el servidor responde con '`401 Unauthorized`' e incorpora la cabecera '`WWW-Authenticate`', pero esta vez además de indicar que el mecanismo de autenticación a usar será de tipo '`Digest`', adjunta un número aleatorio que se utiliza a modo de reto. El cliente repetirá la petición y en esta ocasión incluirá una cabecera '`Authorization`', cuya principal componente es el resumen (por defecto MD5, pero hay una cabecera que permite especificar otro) de identificador de usuario, la contraseña, el número aleatorio, el método HTTP y el URI solicitado. De esta forma la contraseña no se transmite en claro nunca y la captura de un mensaje no compromete la autenticación del siguiente, puesto que el número aleatorio variará de un mensaje a otro.

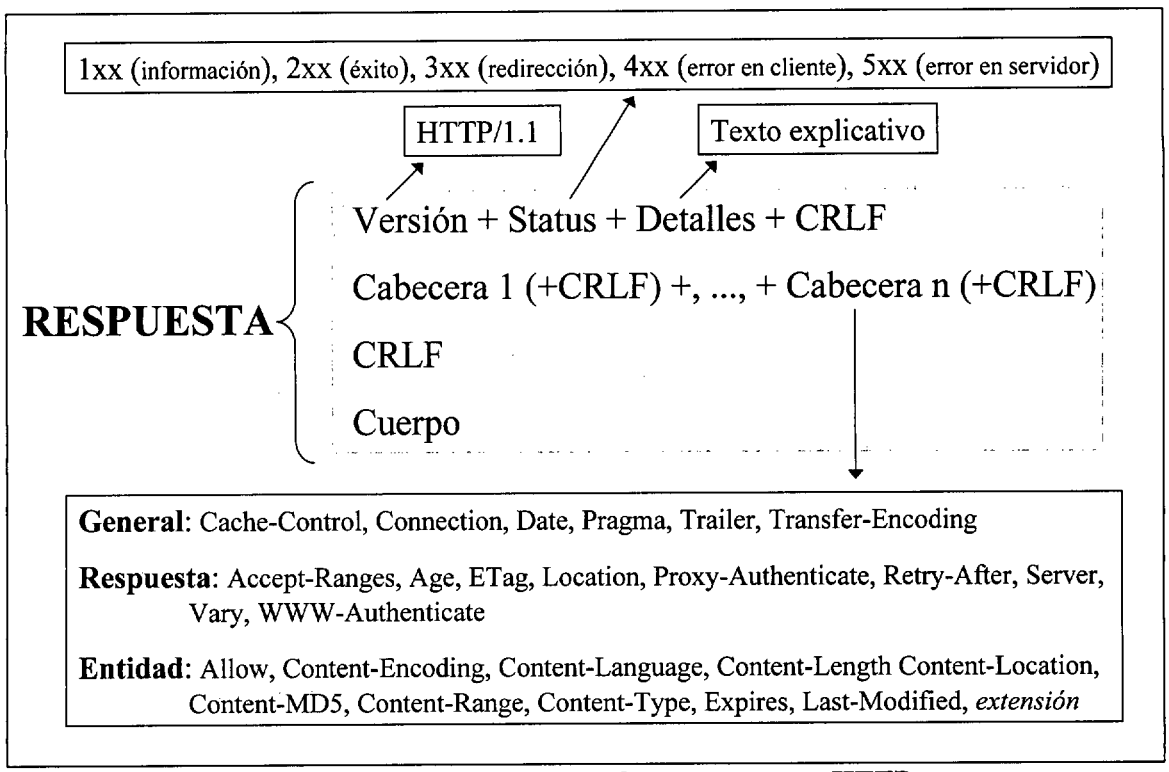


Figura 17. Formato general de una respuesta HTTP

2.3.2.3 HTTP 1.1

Una de las grandes aportaciones de la versión 1.0 era la posibilidad de enviar mensajes haciendo uso de MIME (*Multipurpose Internet Mail Extensions*, [FB96]) con un formato semejante al de los mensajes de correo electrónico lo cual permitía enviar junto con los datos habituales, meta-información sobre dichos datos y modificadores sobre la semántica de los comandos que se invocan.

Sin embargo, esta versión del protocolo tenía una serie de problemas que no se podían dejar de abordar si el objetivo final era la consecución de un protocolo escalable y eficiente capaz de adaptarse a la demanda de recursos que actualmente hay en la red.

En el desarrollo de HTTP 1.1 [Fie+99] se puso un énfasis especial para poder lograr que el protocolo fuese eficiente y no resultase complejo el desarrollo de clientes y servidores.

Las principales características de esta versión del protocolo son:

Conexiones temporales: HTTP es un protocolo que se ejecuta sobre TCP (la especificación en realidad únicamente habla de la necesidad de un protocolo de transporte fiable que se ejecute por debajo y no exige ningún protocolo en concreto ni define la forma de transportar HTTP sobre ese protocolo, pero prácticamente el único protocolo de transporte que se usa en HTTP es TCP). Sin embargo, la versión 1.0 exige la apertura de una conexión TCP por cada URL que se pretende obtener (una conexión diferente por URL), lo cual ocasiona un aumento considerable de la carga en los servidores. Al abrir y cerrar menos conexiones TCP, se ahorra tiempo de CPU en los routers, clientes, servidores, proxies, túneles, caches, etc. y también se ahorra memoria para bloques de control TCP en los ordenadores de los usuarios.

Además los clientes pueden hacer varias peticiones (siempre que sean ídem-potentes como `get`, `head`, `put`, `delete`, `options` o `trace`) sobre la misma conexión sin esperar a obtener una respuesta, con una utilización mucho más eficiente de las conexiones TCP. La congestión de red se reduce pues se reduce el número de paquetes utilizados en aperturas TCP y se deja a las conexiones TCP el tiempo suficiente como para que sean capaces de determinar el estado actual de la congestión de la red (antes se abría una por URL con lo que no se daba tiempo a adaptarse a la situación de la red). No se malgasta tiempo en el procedimiento de apertura TCP, por lo que se reduce el retardo en hacer la petición. Los errores pueden comunicarse sin necesidad de cerrar la conexión TCP. El funcionamiento por defecto de HTTP/1.1 es el de la utilización de conexiones persistentes (el cliente debe asumir que el servidor mantendrá la conexión incluso después de un error por parte del mismo servidor).

Transmisión de mensajes: los servidores deben mantener las conexiones en la medida de lo posible y manejar la congestión en base a los mecanismos de TCP (en vez de cerrar la conexión y esperar a que el cliente retransmita). Al mismo tiempo, los clientes que deben monitorizar la conexión y parar la transmisión si reciben un mensaje de error. Es posible enviar un mensaje al servidor para asegurarse de que el servidor sigue procesando mensajes (se manda un '`Expect: 100-continue`' y se espera un '`100-continue`').

Cache: el mecanismo de caché de la versión 1.0 es muy pobre y en muchas ocasiones no era utilizado ni por clientes ni por desarrolladores de software. Esta nueva versión gestiona mucho mejor todo el tema de tiempo en la caché, expiración de entradas, refresco de caché, obtención de páginas modificadas, mecanismo de avisos (en caso de violación de transparencia semántica, es decir, si la respuesta se estima que no sería la misma que si se hiciese al servidor original), etc.

2.4 Clientes y servidores

2.4.1 Introducción

Hoy en día el concepto de cliente en Internet no se reduce exclusivamente ya al navegador Web. Los posibles usuarios de una plataforma de intermediación pueden variar desde el clásico cliente de HTTP, a un applet que se ejecuta desde una WebTV (pasando por aplicaciones Java, clientes CORBA, etc.).

Sin embargo y puesto que el navegador Web sigue siendo el mecanismo más habitual de acceso (y lo será durante mucho tiempo, tal y como demuestra el hecho de que los navegadores estén dando soporte a tecnologías XML) y que el resto de posibles clientes suelen ser específicos de determinadas arquitecturas de distribución o de servidores de aplicaciones que se verán posteriormente, en este apartado solamente se comentarán los clientes Web.

2.4.2 Clientes

El primer navegador Web gráfico que apareció (capaz de interpretar documentos HTML transmitidos mediante HTTP), lo hizo en el ámbito académico y fue el Mosaic del NCSA (*Nacional Center for Supercomputing Applications*) en 1993. La primera versión sin embargo era tan sencilla, que hasta la versión 2.0 (que en cualquier caso apareció dos meses después) no se consideró el navegador como un producto verdaderamente terminado (implementando ya prácticamente toda la funcionalidad que se vería después recogida en la versión 2.0 de HTML, como por ejemplo, la posibilidad de definir formularios).

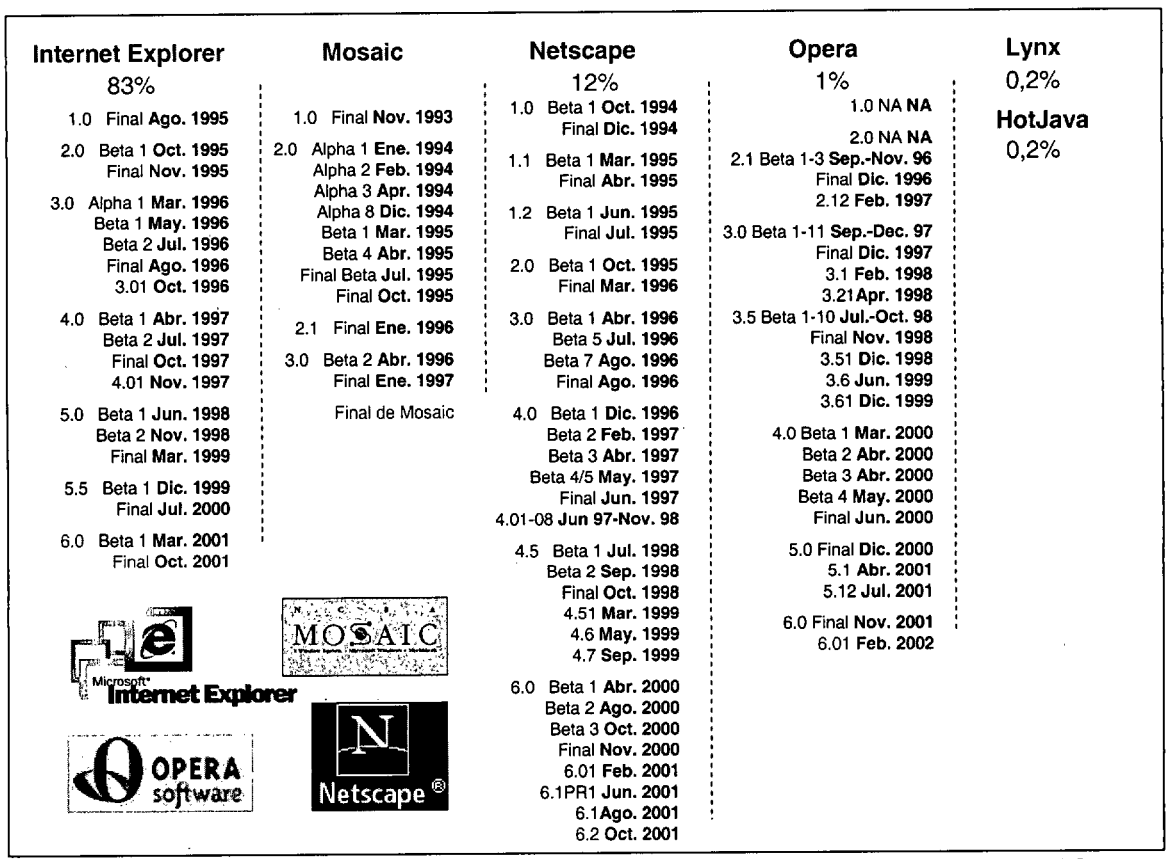


Figura 18. Evolución de navegadores Web y cuota de mercado [BrowserWatch]

A mediados de 1994, el fundador de Silicon Graphics, Jim Clark, junto con Marc Anderssen (director del proyecto Mosaic), crean Mosaic Communications, que posteriormente cambió su nombre por el de Netscape Communications.

A partir de entonces mucha gente del proyecto Mosaic es contratada por Netscape y en pocos meses aparece la primera versión del navegador (ver [Netscape]) y contaba ya con tantos avances tecnológicos que en poco tiempo adquirió el liderazgo del sector: más del 80% de cuota de mercado en 1995.

Con el lanzamiento de Windows95 ese mismo año, apareció también el Internet Explorer de Microsoft [IE], que pese a no ser tan potente como el navegador de Netscape (cuya versión 2.0, supuso además un tremendo avance en el mundo de los clientes Web puesto que ya incluía soporte para Java y Javascript, para marcos y para plugins), contaba con la ventaja de ser gratuito. Poco a poco el Internet Explorer fue acortando la diferencia

tecnológica y para la versión 4 (1998), ambos navegadores eran ya funcionalmente muy similares.

En 1998 Netscape anuncia su intención de distribuir el navegador también de forma gratuita y además anuncia que el desarrollo del código se llevará a un proyecto de código abierto (*open source*), denominado Mozilla [Mozilla] a partir del cual Netscape espera construir su nuevo navegador. Pese a todo, desde este anuncio aún no ha aparecido la versión final 1.0 de Mozilla y las últimas versiones del Netscape (6.x), se basan en versiones específicas de Mozilla (0.6) creadas para dar soporte al navegador, que se separan de la rama oficial del proyecto y cuentan con muchos errores.

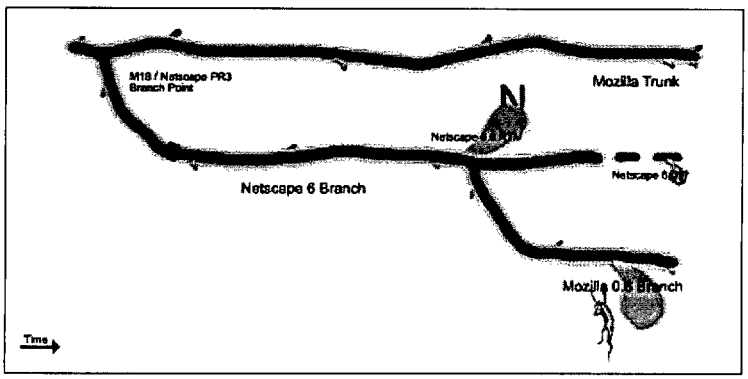


Figura 19. Situación actual de Netscape y Mozilla [Mozilla]

Todos estos detalles y el hecho de que el Internet Explorer haya superado con creces las prestaciones del Netscape y venga ya incorporado en el sistema operativo Windows, ha llevado a que actualmente la cuota de mercado de este navegador esté alrededor del 83% (91% en España [AIMC]).

Independientemente de la velocidad del motor de navegación, los errores que pueda tener en la implementación, etc., la funcionalidad que ofrecen ambos navegadores es prácticamente similar (algo más avanzada la del Internet Explorer, aunque se espera que se vuelvan a igualar en cuanto se concluya Mozilla 1.0):

- Soporte para las últimas versiones de HTML y HTTP.
- Soporte para diferentes tecnologías de descripción, visualización de datos y comunicación: DHTML, CSS, DOM, RDF, Javascript, Java, SMIL, .NET, CORBA, etc.
- Seguridad: soporte para diferentes mecanismos de seguridad (gran control sobre detalles de privacidad, firma digital, cifrado de datos, soporte para protocolos seguros de transmisión, etc.).
- Interfaces de usuario muy sofisticadas.
- Integración con utilidades de correo electrónico, canales de *chat*, herramientas de desarrollo Web, utilidades de audio y video, herramientas de telefonía, etc. (en línea con el éxito del Internet Explorer, en España el 65% de los internautas utiliza Outlook para leer el correo electrónico, frente al 13% que utiliza el Netscape Messenger [AIMC]).

2.4.3 Servidores

2.4.3.1 Servidores Web

El mismo éxito que obtuvo el NCSA con su navegador Mosaic, lo repitió con su servidor Web, uno de los primeros en aparecer en Internet. Sin embargo el soporte para el desarrollo de dicho servidor cesó a mediados de 1994, cuando el máximo responsable del proyecto dejó el NCSA (posteriormente se retomó, aunque no con tanto éxito).

En Abril de 1995, aparece la versión 0.6.2 de Apache que está basado en la 1.3 del servidor del NCSA (código abierto) a la cual se le habían solucionado gran cantidad de problemas que presentaba y se le habían añadido gran cantidad de mejoras [Apache]. Las sucesivas mejoras que van apareciendo revisiones de este servidor (siempre gratuito), hacen que en menos de un año supere en penetración al servidor del NCSA, colocándose como líder en el sector y manteniendo dicha posición hasta el momento (Figura 20).

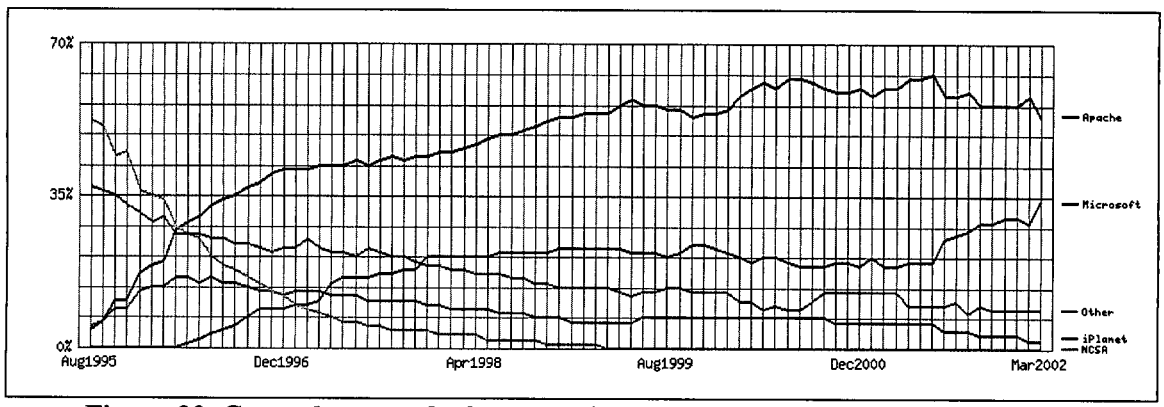


Figura 20. Cuota de mercado de los servidores Web más importantes [Netcraft]

A la funcionalidad básica de atender peticiones de ficheros a través del protocolo HTTP, localizar dichos ficheros y devolvérselos al cliente mediante el mismo protocolo, pronto se le añadió la de soporte para CGI (1993) que propició el primer mecanismo dinámico de comunicación cliente-servidor en el Web (permite que el cliente envíe datos al servidor).

El resto del apartado se centrará en las posibilidades que ofrecen los servidores precisamente desde este punto de vista de interacción con los clientes.

2.4.3.2 CGI

2.4.3.2.1 Introducción

El mecanismo (protocolo) CGI (*Common Gateway Interface,* [CGI01]), permite generar contenido Web (páginas HTML, sonidos, imágenes, etc.) dinámicamente, basándose en información obtenida de formularios, listas de selección, botones, etc.

Como ya se ha visto, el protocolo HTTP permite que desde el cliente se pueda pedir un fichero al servidor, éste lo localice y lo envíe de vuelta al cliente. La idea de CGI es que el servidor se configure de tal forma, que si el fichero solicitado reside un

determinado directorio, en lugar de enviárselo al cliente lo que se hace es ejecutarlo como un programa (pasándole como parámetros de entrada los datos que envía el cliente) y lo que se devuelve al cliente es la salida de dicho programa.

Pese a que es ya un mecanismo un tanto obsoleto (aparece en 1993), se resiste a desaparecer por su gran simplicidad y versatilidad y por ser totalmente compatible con todas las plataformas. Sin embargo, CGI tiene algunas limitaciones que poco a poco lo van desplazando, en beneficio de nuevas tecnologías como pueden ser los servlets o las páginas de servidor de Java (JSP).

2.4.3.2.2 Funcionamiento CGI

A partir de una página del estilo de la que se aprecia en la Figura 21, el servidor recibe los datos del cliente, localiza el fichero CGI (`prueba.cgi` en el ejemplo), lo ejecuta y además le pasa los datos que el cliente haya podido introducir en los formularios.

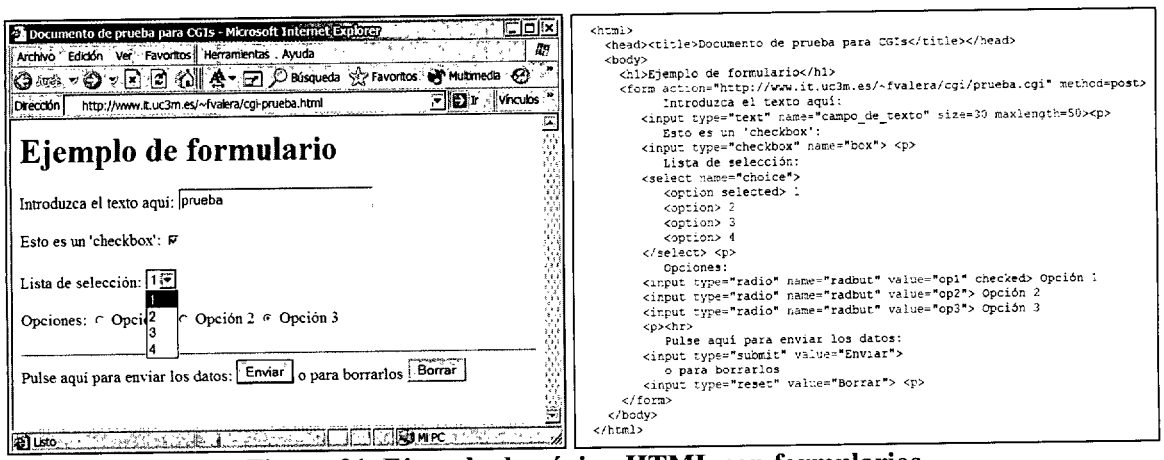


Figura 21. Ejemplo de página HTML con formularios

En función de que el método de transmisión que haya asociado al botón con la orden '`submit`' en la página HTML sea GET o POST, los datos del cliente se transmiten de una forma u otra:

- ***GET***: la información del usuario se incorpora después de una '`?`' a la URL que le llegará al servidor y éste se la pasa al programa CGI como la variable de entorno `QUERY_STRING`. De escoger este método, en el ejemplo anterior se enviaría:

```
http://www.it.uc3m.es/~fvalera/cgi/prueba.cgi?campo_de_texto="prueba"&box="1"&choice="1"&radbut="opt1"
```

- ***POST***: la información le llega al programa CGI por la entrada estándar con lo que puede ser leída con los mecanismos de entrada-salida habituales (en vez de en la URL, se transmite con el cuerpo del mensaje HTTP).

Finalmente la respuesta del programa (una página HTML por ejemplo) es devuelta al cliente utilizando HTTP e interpretada por el navegador que generó la petición.

2.4.3.2.3 Problemas del modelo CGI

El modelo CGI ha sido ampliamente utilizado por su sencillez a lo largo de muchos años, pero siempre ha estado expuesto a las críticas por determinados problemas o

limitaciones que presenta y que hoy en día están haciendo que la funcionalidad que ofrecían los CGIs sea asumida por otro tipo de tecnologías más modernas.

Uno de los principales problemas que presentan los CGI es la velocidad de ejecución. No sólo es ya cuestión de que en muchos casos el código no esté compilado y deba ser interpretado (caso de Perl y resto de lenguajes de scripting, por ejemplo). El modelo de ejecución de un CGI exige que con cada petición del cliente (ver Figura 24), independientemente de que ésta sea igual o distinta de la anterior, el servidor Web se encargue de localizar al CGI responsable de atender dicha petición y de iniciarlo. Pese a que este modelo puede funcionar en el caso de peticiones sencillas o limitadas en número y tamaño, lo cierto es que no termina de encajar en los esquemas escalables que hoy en día se tiende a desarrollar.

Por otro lado, los programas CGI no mantienen el estado de las conexiones con los clientes y por tanto no guardan datos entre una petición y otra. Esto limita mucho la funcionalidad que pueden ofrecer estos programas.

Por último, desde la óptica arquitectural hoy en día se tiende a separar claramente las funciones que realiza cada componente. En el caso de los CGI la presentación final de los datos y la lógica de negocio del programa están mezcladas y eso hace que no sea la solución óptima para el modelo de arquitectura modular y flexible que ya se demanda.

2.4.3.3 Servlets

Los servlets [Servlet] son la respuesta de la tecnología Java a los programas CGI y al igual que ellos son programas que residen en los servidores y generan dinámicamente páginas Web. Las principales ventajas con respecto a los CGIs son:

- ***Eficiencia***: en los programas CGI, cada vez que llega al servidor una petición HTTP se inicia un proceso diferente, con lo que si el programa se ejecuta con rapidez, es posible que el tiempo que se emplea en iniciar el programa, sea incluso mayor que el que se tarda en ejecutarlo. En el caso de los servlets, el único proceso activo es el que está ejecutando la máquina virtual y cada llamada es atendida por un *thread* de Java, mucho más ligero que un proceso del sistema operativo (ver Figura 24). Además en vez de replicar el código del programa en memoria con cada invocación, en los servlets sólo existe una copia.
- Los servlets están específicamente diseñados para el ***tratamiento automático*** de datos HTML y para la generación de páginas HTML de respuesta.
- Los servlets permiten llevar a cabo muchas ***más acciones*** que los CGIs (conectarse directamente con el servidor Web, compartir información entre diferentes servlets, mantener información de petición en petición, enviar información a diferentes EJBs, etc.).
- Están escritos en Java siguiendo un API estándar, por lo que pueden ser ***portados*** sin ningún cambio a cualquier servidor Web que soporte servlets (y prácticamente todos los servidores conocidos los soportan bien directamente o con algún *plugin*).
- Ventajas de ***seguridad***: al igual que los applets, están diseñados para ejecutarse dentro de un entorno de funcionalidad restringida (*sandbox*), lo cual proporciona un mayor control de las posibles acciones del servlet sobre recursos sensibles del sistema.

La página HTML de entrada es similar a la de los CGIs, con su correspondiente botón de `submit` y el correspondiente método GET o POST, pero ahora se incluye un identificador del servlet detrás del método elegido en la etiqueta FORM (se añade `ACTION="nombre_del_servlet"`).

El servlet generalmente es una clase Java que implementa la interfaz `javax.servlet.Servlet` y entiende la versión 1.1 del protocolo HTTP. Para cada método HTTP (`GET`, `HEAD`, `POST`, etc.) hay un método `do` en el servlet que es capaz de gestionarlo y que recibe como parámetros de llamada los datos de la petición.

Igual que en el caso del CGI, lo que el servlet escribe en la salida estándar, se le transmite al cliente como respuesta.

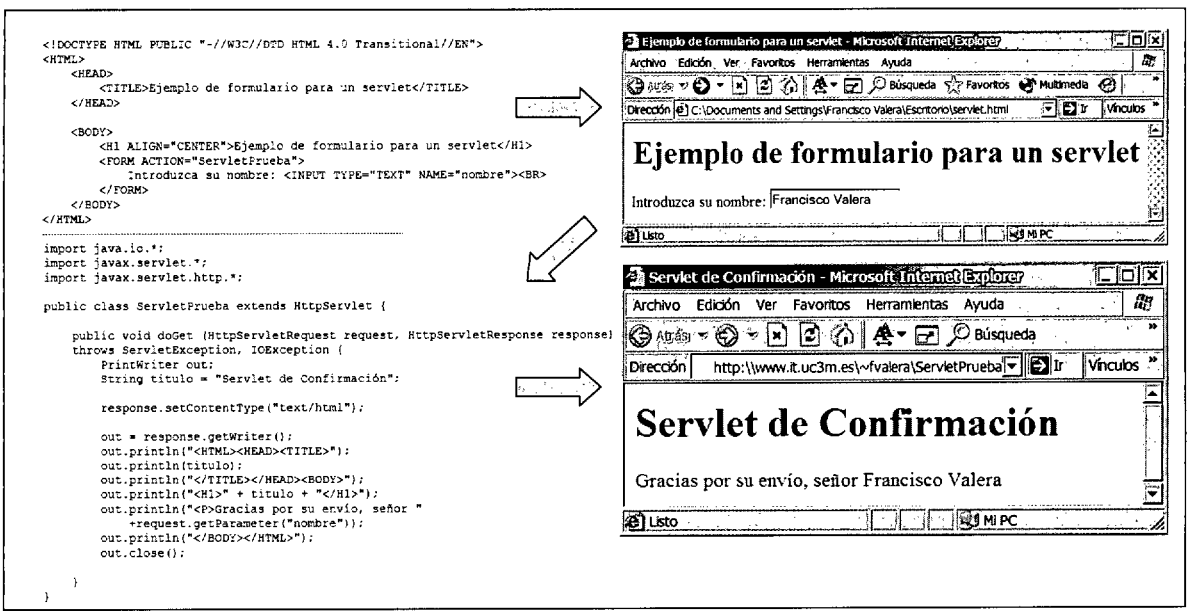


Figura 22. Ejemplo de petición y respuesta con servlets

2.4.3.4 JSP

La tecnología JSP (*JavaServer Pages*, [JSP]) permite separar la lógica de la generación dinámica, de las páginas HTML estáticas (los CGIs y los servlets generan páginas dinámicas con cada petición del usuario cuando en la mayoría de los casos las variaciones entre las páginas generadas son mínimas).

Los ficheros JSP son documentos con el formato de las páginas HTML, pero incluyen llamadas a métodos que permiten generar la parte dinámica de la respuesta (como *scripts* de servidor, tienen una filosofía muy similar a las ASPs, *Active Server Pages de Microsoft* [ASP], aunque si bien las JSPs son portables a cualquier tipo de servidor, las ASPs están restringidas a las plataformas Windows). Dichos documentos se ubican donde normalmente se ubicarían las páginas Web y son transformados de forma automática a servlets para tratar la parte dinámica.

En principio las JSP no pueden hacer nada que un servlet no haga, pero la ventaja de separar el contenido de la lógica de presentación es que la persona que se encarga de actualizar el aspecto de las páginas no tiene por qué entender el código Java y de la misma forma, la persona que actualiza el código Java no tiene por qué ser un experto en

el diseño de páginas Web. Además resulta mucho más sencillo modificar un documento de texto HTML que una serie de invocaciones 'println' en una clase Java (y luego volver a recompilar).

Tanto las JSP como los servlets ofrecen gran cantidad de ventajas sobre los CGIs y es por eso que están empezando a desplazarlos en los esquemas actuales de interacción: independencia de la plataforma de ejecución (por estar basados en Java), menor sobrecarga (ver Figura 24), permiten acceder de forma sencilla a los parámetros que se pasan del cliente al servidor, ofrecen mejores posibilidades de conexión con servidores de aplicación o terceras aplicaciones en general, etc.

2.4.4 Evolución de los esquemas de clientes y servidores

La Figura 23 resume de forma gráfica la evolución de los diferentes esquemas de interacción entre clientes y servidores.

Los primeros esquemas estaban basados principalmente en navegación Web y eran por tanto dependientes de las posibilidades de los navegadores que solían limitarse a la visualización de imágenes. Desde el punto de vista del comercio electrónico, la única utilización que se le daba era prácticamente la promoción y el marketing de empresas y productos.

Posteriormente aparecieron los modelos basados en CGIs, capaces poco más que de generar respuestas en función de los formularios HTML procedentes de los navegadores de los clientes, que en su progresiva evolución eran ya capaces de procesar audio y video.

La siguiente etapa se alcanzó tras un considerable esfuerzo por potenciar las capacidades de los clientes trasladando en algunos casos la lógica de las aplicaciones a los navegadores que podían procesarla gracias a las máquinas virtuales que ejecutaban applets y la posibilidad de interpretar scripts de Visual Basic o JavaScript.

Durante varios años, los applets se consideraron como la herramienta principal para crear aplicaciones Web basadas en Java y por tanto la clave para la generación de un esquema de servicios Web potente y flexible. Sin embargo, actualmente dicha línea de desarrollo se ha ido abandonando poco a poco por diferentes motivos:

- ***Problemas de*** compatibilidad ***con los navegadores***: desde que los navegadores Web incorporaron las primeras implementaciones de máquinas virtuales, se empezaron a plantear diferentes problemas de compatibilidad con los applets que se programaban siguiendo las especificaciones de referencia de Sun. La mayor parte de los problemas vinieron generados por errores de las máquinas virtuales o porque las implementaciones no terminaban de cubrir todas las APIs que estaban especificadas. Todo ello llevó a que Netscape dejase de seguir actualizando las versiones de las máquinas virtuales y comoquiera que el Internet Explorer no terminaba de dar soporte a Java, actualmente Sun se encarga de desarrolla un plugin que proporciona junto con cada implantación que saca al mercado. Dicho plugin debe necesariamente ser descargado del Web de Sun para que el cliente adquiera toda su potencia.

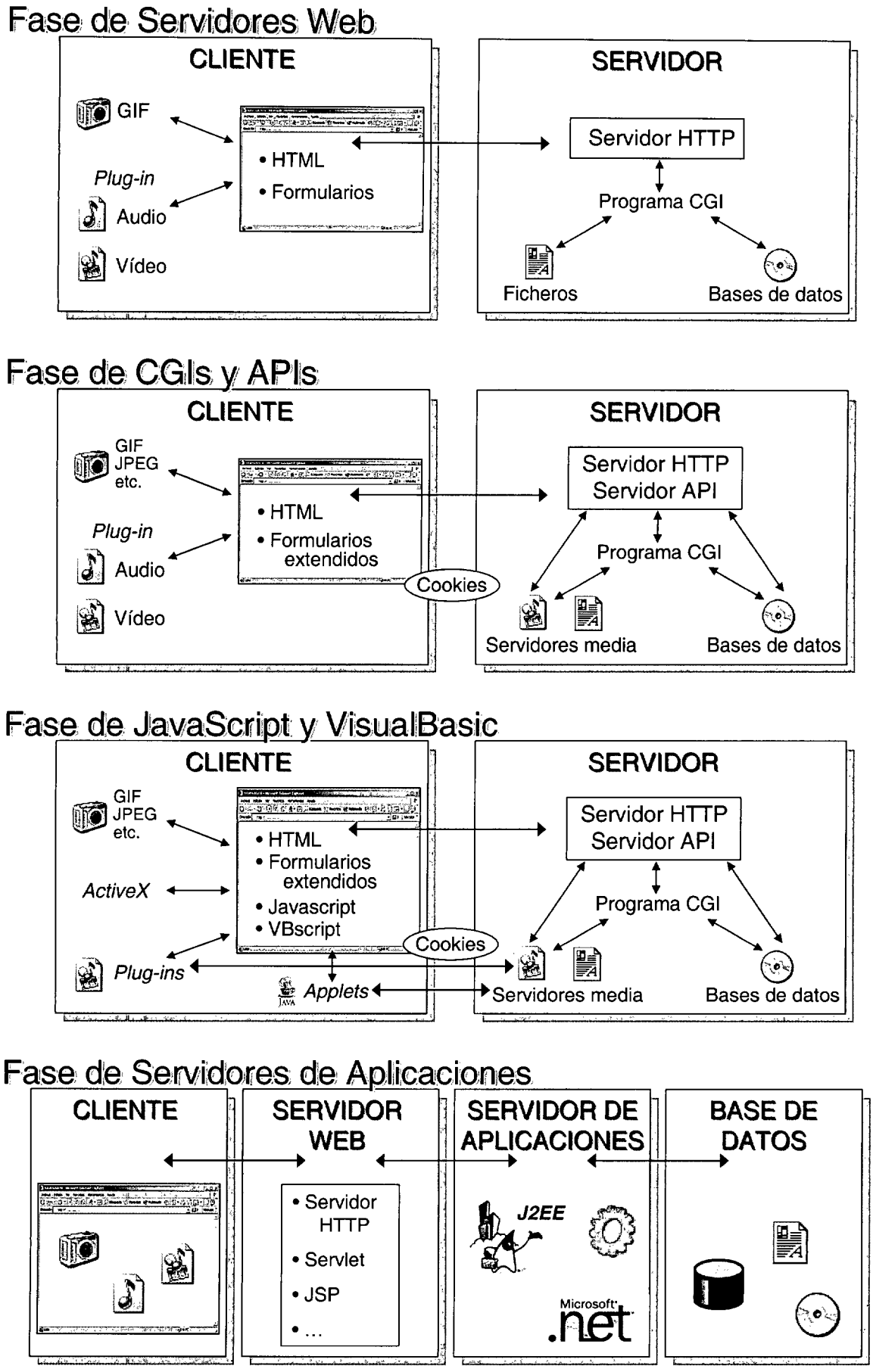


Figura 23. Evolución de los esquemas de interacciones Web

- Las restricciones ***de seguridad*** impuestas por los navegadores (*sandbox* de la máquina virtual), acotan mucho las posibilidades de los applets tanto desde el punto de vista de los clientes como desde el punto de vista de los desarrolladores.
- El tiempo de descarga de los applets y de las clases asociadas (un ORB, por ejemplo), si bien con el ***ancho de banda*** actual se ha rebajado considerablemente, años atrás no era ni mucho menos despreciable.
- El tiempo que tarda la ***máquina virtual*** en iniciarse hace que la navegación por páginas que incorporen applets sea considerablemente incómoda.
- La carga ***y descarga*** de applets y el inicio y terminación de la máquina virtual hacen que los navegadores entren en situación inestable con demasiada frecuencia.
- El procesamiento llevado a cabo en el ***cliente*** cuando se exige acceso a bases de datos, conexiones con servidores externos, sobrecarga en ocasiones los ordenadores de los clientes.

Mientras tanto, en los servidores se ampliaron las posibilidades de los CGIs haciendo que pudiesen acceder de forma más transparente y rápida a bases de datos y diversos servidores con recursos multimedia y posibilitando la utilización de muchos más lenguajes (diferentes de C o Perl) para la implementación de los programas CGI.

El siguiente gran paso en la evolución de los esquemas de interacción, fue la aparición de la tecnología de los servlets. Estos se ejecutan completamente en el servidor por lo que no están sujetos a problemas de compatibilidad con los navegadores clientes o a problemas con el tiempo de descarga, etc. Además las facilidades que ofrecen en cuanto a desarrollo y en cuanto a la rapidez en la ejecución que aportan los diferentes hilos de ejecución de Java, han llevado a muchas empresas a moverse de los modelos basados en CGI a los modelos basados en servlets.

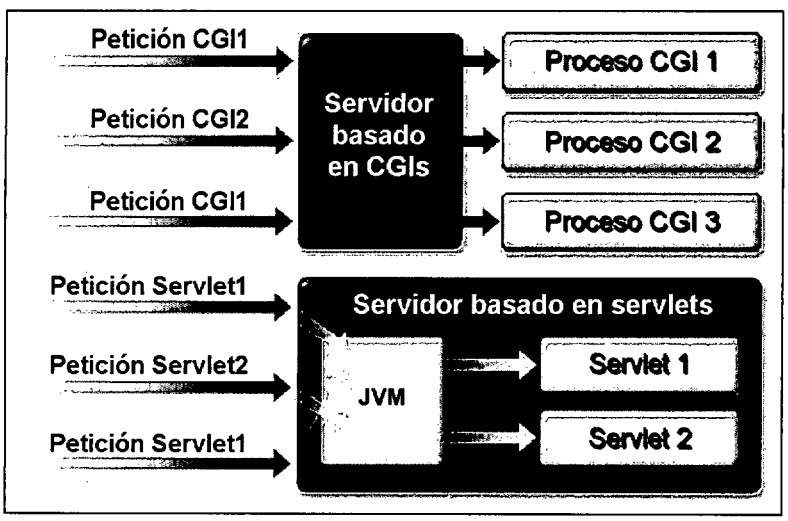


Figura 24. Comparación del modelo de ejecución de CGIs y Servlets

Poco después aparecieron las JSP, que culminaron la evolución de los servlets permitiendo que estos fuesen programados de una forma mucho más sencilla, a base de patrones integrables en páginas HTML que finalmente se compilaban y se convertían en servlets.

Actualmente se observa una tendencia en los esquemas de interacción Web, hacia una arquitectura escalable en la que están descentralizadas las tareas que antes estaban

exclusivamente atribuidas a los servidores Web. Hoy en día la línea de desarrollo se mueve hacia estas arquitecturas multicapa, (ver capítulo 3) en las que los servidores de aplicaciones juegan un papel muy importante haciendo de capa *middleware* y ubicándose entre el cliente (navegadores, clientes potenciados por applets de Java, aplicaciones Java, dispositivos inalámbricos, etc.) y la capa de datos (normalmente bases de datos relacionales o cualquier otro tipo de fuente de información empresarial), típicamente después de los servidores Web, asumiendo la carga de la lógica de negocio que antes era asumida por el cliente, el servidor Web o los servlets.

Se pretende conseguir así un sistema modular en el que estén separados por un lado el cliente, por otro la capa Web responsable de atender las peticiones del cliente, por otro la capa que gestiona la lógica de negocio y que independiza ésta por completo de los detalles relacionados con el acceso de los clientes y por último la capa de datos, que se encargaría de gestionar toda la complejidad inherente al acceso a las bases de datos.

2.5 Servicios Web

La tendencia actual de los modelos de clientes y servidores, potenciados por todas las tecnologías de distribución que se verán en el capítulo siguiente, está orientada a la provisión de lo que se ha dado en llamar 'servicios Web'.

Dichos servicios ofrecen un nuevo esquema de aplicaciones Web. Como tales, son modulares, autocontenidas y autodescriptivas y pueden ser publicadas, localizadas e invocadas de forma transparente a través del Web. Las funciones que desempeñan pueden ir desde la resolución de peticiones simples a la ejecución de complicados modelos de negocio. Una vez que un servicio Web se despliega, otras aplicaciones (y otros servicios Web) pueden descubrir e invocar dicho servicio automáticamente.

Pese a todo lo que la definición anterior pueda sugerir, en el fondo estos servicios Web no son sino una serie de estándares tecnológicos, que pretenden dar soporte a todo el avanzado modelo de negocio que se ha estructurado a su alrededor (ver Figura 25).

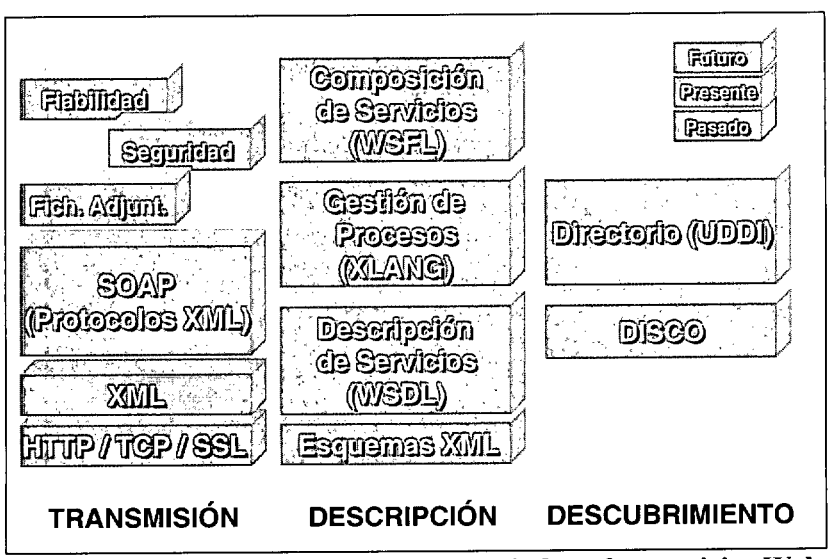


Figura 25. Conjunto de tecnologías asociadas a los servicios Web

El acceso a un servicio Web por parte de un usuario (o de otro servicio Web), se traduce al final en una invocación de un determinado método (o una serie de métodos) en el proveedor que ofrece dicho servicio Web. Sin embargo, antes de que una entidad pueda enviar el correspondiente mensaje SOAP requiriendo la invocación del servicio Web, hay que localizar el negocio en cuestión capaz de proporcionar el servicio que se necesita.

El mecanismo más difundido para solucionar el problema de la publicación y la búsqueda de servicios Web es UDDI (*Universal Description, Discovery and Integration*), que permite dirigir a los sistemas que estén buscando dichos servicios hacia la información que los describe (ver [UDDI]). UDDI utiliza mensajes SOAP para publicar, editar, visualizar o buscar información en un registro.

Una vez localizado el servicio en el registro, es necesario obtener la interfaz de acceso y su semántica asociada para poder interoperar con él. El lenguaje WSDL (*Web Services Description Language*, [WSDL]), permite escribir documentos XML que describan todos estos detalles que se necesitan conocer de un servicio (dichos documentos son los que estarán almacenados en registros UDDI).

En el futuro se espera que las líneas de investigación actuales sobre servicios Web (y que incluyen tecnologías como las que se pueden ver en la Figura 25) y la denominada Web semántica (ver [BHL01] y [SW]), que pretende proporcionar una extensión al modelo actual del Web en el que la información no carezca de un significado y éste pueda ser interpretado por los ordenadores, converjan de tal forma que se llegue a un modelo de servicios avanzados capaces de obtener de Internet todo su potencial como soporte de servicio Web inteligentes (Figura 26).

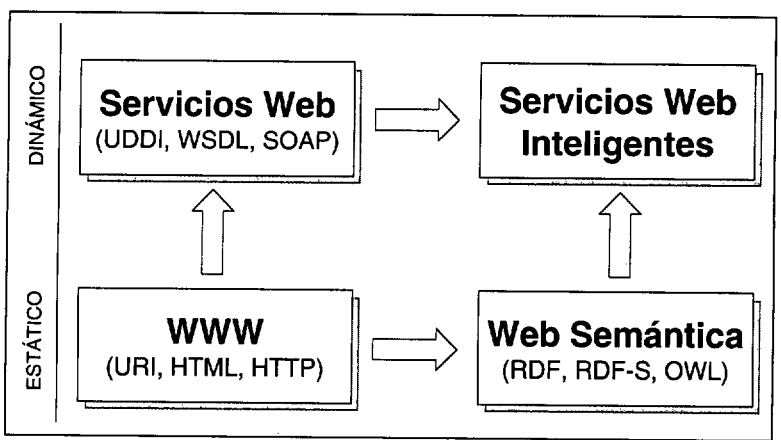


Figura 26. Convergencia entre la Web semántica y los servicios Web

2.6 Conclusión

A lo largo de este capítulo se han ido presentado diferentes tecnologías denominadas como '*básicas*', puesto que son el soporte en el que se fundamentan los mecanismos de distribución que se verán en el siguiente capítulo y por extensión las arquitecturas que a su vez utilizan dichos mecanismos.

Con el objetivo de fijar criterios que permitan abordar el resto de la tesis con una perspectiva amplia acerca de las implicaciones que tiene la elección de una u otra tecnología, se han aportado valoraciones comparativas sobre cada una de ellas y además se han examinando sus respectivas evoluciones temporales de forma que sea posible razonar de forma más sencilla sobre la situación actual de todas ellas.

En línea con estos comentarios, se ha visto como el lenguaje Java, pese a contar con una madurez muy inferior a la de sus más serios competidores C y C++ (más de una década) y recibir gran cantidad de acusaciones (fundadas) referentes a su mal rendimiento, se ha convertido en un estándar de facto de la programación actual en Internet. A lo largo del año 2002, el número de programadores de Java superará al de C y C++ y con respecto a las prestaciones, si bien no llegará tan lejos, sí que será prácticamente equiparable a estos otros lenguajes.

Con respecto a su funcionalidad como lenguaje para potenciar las capacidades de los clientes Web (mejora del entorno gráfico, adquisición de responsabilidades en la lógica de negocio, etc.), lo cierto es que aunque es una característica que llegó incluso a justificar la existencia del lenguaje, hoy en día está cayendo en desuso en beneficio de los denominados *clientes ligeros* (*thin clients*).

La tendencia actual va más orientada a la utilización de clientes rápidos basados en un navegador Web (el Internet Explorer de Microsoft se ha impuesto ya al Navigator de Netscape sin ningún género de dudas) que se conectan a los servidores para obtener de ellos documentos de texto estructurado que saben representar y que cada vez más, se aproximan al modelo de documentos especificado por el conjunto de tecnologías XML. Dicho lenguaje, va constituyéndose progresivamente en el lenguaje de representación e intercambio de datos de Internet.

La verdadera proyección de Java hoy en día (habiendo por otro lado, solucionado ya los problemas de integración con los navegadores que empezó a presentar a medida que la máquina virtual ganaba en complejidad, e incluso potenciando la presencia de Java en el cliente con tecnologías como Java WebStart™) está en el campo de los servidores.

Ya desde hace unos años, la tecnología Java empezó a aportar alternativas al rígido esquema de interacción con el cliente que ofrecía el mecanismo CGI. Primero con los servlets y luego con las páginas de servidor (JSP), Java se introducía de lleno en el dominio de los servidores con un esquema tremendamente potente y muy versátil.

El paso definitivo para la implantación de Java en el dominio del servidor, tal y como se verá en el siguiente capítulo, se dará con la integración en una misma plataforma (el servidor de aplicaciones J2EE) de gran cantidad de APIs para el desarrollo de aplicaciones de empresa y de servicios Web, así como de un completo entorno de ejecución.

Capítulo 3

Tecnologías de distribución de objetos

3.1 Introducción

Uno de los primeros y más importantes mecanismos de comunicación entre aplicaciones basado en el *paradigma cliente/servidor*, consiste en utilizar la interfaz *socket* directamente para enviar y recibir datos entre los dos extremos de la transmisión. Todos estos datos deben no obstante estar formateados u ordenados de una manera prefijada para que puedan ser entendidos por ambos contertulios. Sin embargo, cuando este concepto intenta generalizarse en un mundo de componentes tan heterogéneo como es Internet, lo cierto es que la labor se torna muy complicada, puesto que cliente y servidor pueden en principio estar programados en diferentes lenguajes o estar ejecutándose en diferentes sistemas operativos, bajo diversas arquitecturas *hardware* y ubicados en diferentes redes.

La idea principal que introduce la computación distribuida en el sentido en el que se tratará aquí, es precisamente la de ser capaz de dividir los procesos de una aplicación entre dos o más sistemas conectados en red, ejecutando métodos de forma remota si es necesario, sin tener que preocuparse de la ubicación de los componentes en la red o de la plataforma de ejecución. La computación distribuida permite aprovechar las ventajas de diferentes plataformas y compartir información entre ellas tratando de hacer transparentes los detalles de bajo nivel como la gestión de *sockets*.

Uno de los logros más importantes en las implementaciones y diseños de sistemas distribuidos, fue la creación de un método homogéneo para intercambiar información entre los diferentes módulos participantes. Este mecanismo común se denominó *middleware* (capa intermedia) y es responsable de conectar las diferentes partes de la aplicación distribuida garantizando transparencia tanto desde el punto de vista de los desarrolladores de aplicaciones, como desde el punto de vista de ubicación de cada módulo o de los sistemas operativos o plataformas software que los sustenten.

Por otro lado, se tiene el *paradigma de orientación a objetos*, tecnología que es hoy en día muy utilizada a la hora de diseñar y construir componentes software por la gran cantidad de ventajas que típicamente se le atribuyen (gestión sencilla, reutilización, herencia, modularidad, encapsulación, etc.).

Como resultado de la unión de ambas tecnologías, la computación distribuida de objetos incrementa las posibilidades de un sistema programado siguiendo las ideas de la orientación a objetos y permitiendo que los objetos que van a ser distribuidos en un entorno de red heterogéneo interactúen de forma global [Sur98]. Y más aún, el principal objetivo de los sistemas de distribución de objetos es permitir que las aplicaciones interactúen con los objetos remotos de la misma forma que si fuesen locales (de manera transparente tanto para los clientes como para los programadores de aplicaciones).

*El **objetivo de este capítulo** es presentar los esquemas middleware con orientación a objetos que se empezaron a utilizar a mediados de los años 90 y que actualmente están sirviendo de base para herramienta de nivel superior y de reciente implantación en Internet: los servidores de aplicaciones. Estas tecnologías constituyen el núcleo de las arquitecturas multicapa que se presentarán al final del capítulo y entre las cuales hoy en día destacan principalmente dos especificaciones J2EE y .NET.*

3.2 Middleware

Los productos que hoy en día reciben la denominación de *middleware*, son infraestructuras de software necesarias para la integración de diferentes aplicaciones y componentes en entornos distribuidos y heterogéneos. Estos productos ocultan la mayor parte de la complejidad inherente a la resolución de dicha heterogeneidad y son los responsables de dotar a la arquitectura software de los diferentes grados de transparencia que se han comentado en capítulos anteriores.

Además de está funcionalidad básica que generalmente lleva asociada la resolución de las comunicaciones de red, las plataformas middleware suelen ofrecen servicios que normalmente no es posible ofrecer en un solo sistema como nombrado, localización de servicios, ejecución remota transparente, transferencia de datos, notificación de eventos, sincronización, servicios de seguridad, etc.

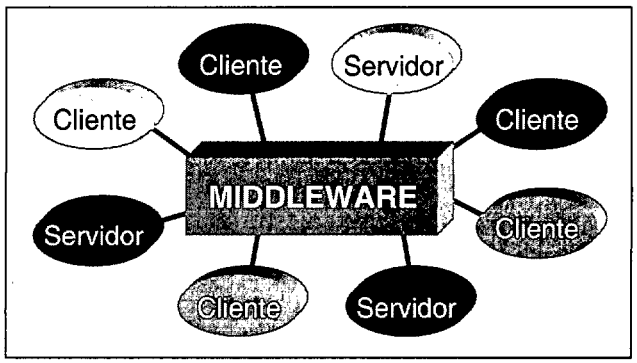


Figura 27. Paradigma de plataforma middleware

Al desarrollar aplicaciones distribuidas, es deseable no sólo que sean capaces de interoperar entre ellas basándose en la plataforma middleware sino que también sean capaces de comunicarse con otras aplicaciones asociadas quizá a otra plataforma diferente. Algunos mecanismos de comunicación como RPCs, middleware orientado a mensaje o middleware basado en procesamiento de transacciones distribuidas o en bases de datos permiten que las aplicaciones interoperen a través de diferentes plataformas aunque normalmente se basan en capas middleware propietarias.

Algunas de las plataformas más populares que se analizan aquí (orientadas a objetos, para obtener los beneficios de dicha aproximación), intentan manejar la diversidad del problema utilizando protocolos comunes para normalizar las comunicaciones de red.

3.3 Arquitectura Común

Aunque cada plataforma tiene su propia denominación y diferente funcionalidad según los componentes en los que esté basada, la mayoría utiliza un esquema semejante para ofrecer transparencia en la distribución de objetos (ver Figura 28)

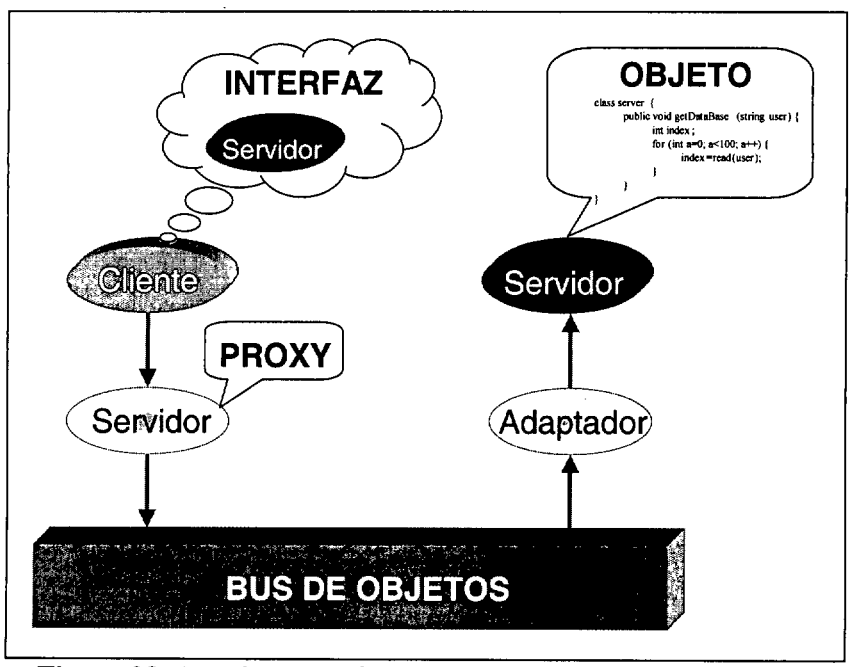


Figura 28. Arquitectura típica de una plataforma middleware

Cuando un cliente quiere contactar con un objeto servidor lo hace utilizando una referencia (software) a dicho objeto. Esta referencia es un *puntero* al objeto, que permite que las peticiones para realizar operaciones y acceder a los datos se transmitan automáticamente por un bus de objetos (el llamado ORB u '*Object Request Broker*' en CORBA), que es capaz de detectar si el objeto destinatario está ubicado en la misma máquina o en otro ORB que se esté ejecutando en una máquina distinta. La referencia identifica el objeto de manera única, ofreciendo información sobre la máquina en la que reside el objeto y la interfaz que el objeto soporta.

Cada objeto del bus debe tener su correspondiente implementación, que es el código (teóricamente en cualquier lenguaje de programación) que especifica cómo llevar a cabo la funcionalidad atribuida al objeto en la máquina correspondiente (apoyado en el lenguaje, sistema operativo y hardware). Cuando una petición alcanza finalmente el bus donde reside la implementación, se transfiere a un adaptador, que es el nexo de unión entre la implementación del objeto y su presencia en el bus.

Para poder saber qué tipo de peticiones pueden hacerse sobre un determinado objeto servidor y motivado también por el hecho de que en general estos servidores pueden estar escritos en cualquier lenguaje, es necesario definir una interfaz para establecer los métodos, atributos y excepciones que el cliente tendrá accesibles (preferiblemente en un lenguaje neutral). A partir de esa interfaz, unos compiladores especiales generarán un código específico (cabos y esqueletos) en el lenguaje elegido para implementar el objeto por los desarrolladores, que ocultará totalmente la tarea de la comunicación entre objetos. Estos cabos se utilizarán en el proceso cliente para comunicarse con el servidor: serialización (*marshalling*), des-serialización (*unmarshalling*), enviar y recibir peticiones y resultados. Por eso, desde el punto de vista del cliente aparecen como un proxy y las llamadas al objeto servidor se hacen en local a dicho proxy.

3.4 Common Object Request Broker Architecture

3.4.1 Introducción

Tras su creación en 1989 el OMG (*Object Management Group,* [OMG]) una de las primeras contribuciones de este grupo fue la especificación de una arquitectura genérica de gestión de objetos, la OMA (*Object Management Architecture*), que constituida a partir de cinco componentes básicos (ver Figura 29):

- ***Objetos de aplicación***: se definen específicamente para aplicaciones concretas, no están estandarizados por el OMG y se basan en los otros cuatro componentes.
- ***Interfaces de dominio (domain interfaces)***: similares a los anteriores, pero orientados a sectores más genéricos de mercado (salud, finanzas, comercio electrónico, telecomunicaciones, etc.).

- ***Facilidades comunes (common facilities)***: incluyen servicios que pueden compartir muchas aplicaciones e interfaces de dominio. Hay facilidades horizontales y verticales según se dirijan a diferentes sectores o se utilicen dentro del mismo nivel.
- ***Bus de objetos***: permite interconectar todos los objetos (de aplicación, interfaces de dominio, facilidades comunes y servicios de objetos) de manera transparente.
- ***Servicios de objetos***: son interfaces y objetos de carácter general que constituyen la base de prácticamente cualquier aplicación. Tanto las facilidades comunes (también de carácter general, pero no tan básicas como estos servicios), como las interfaces de dominio o los objetos de aplicación están sustentados por estos servicios de objetos.

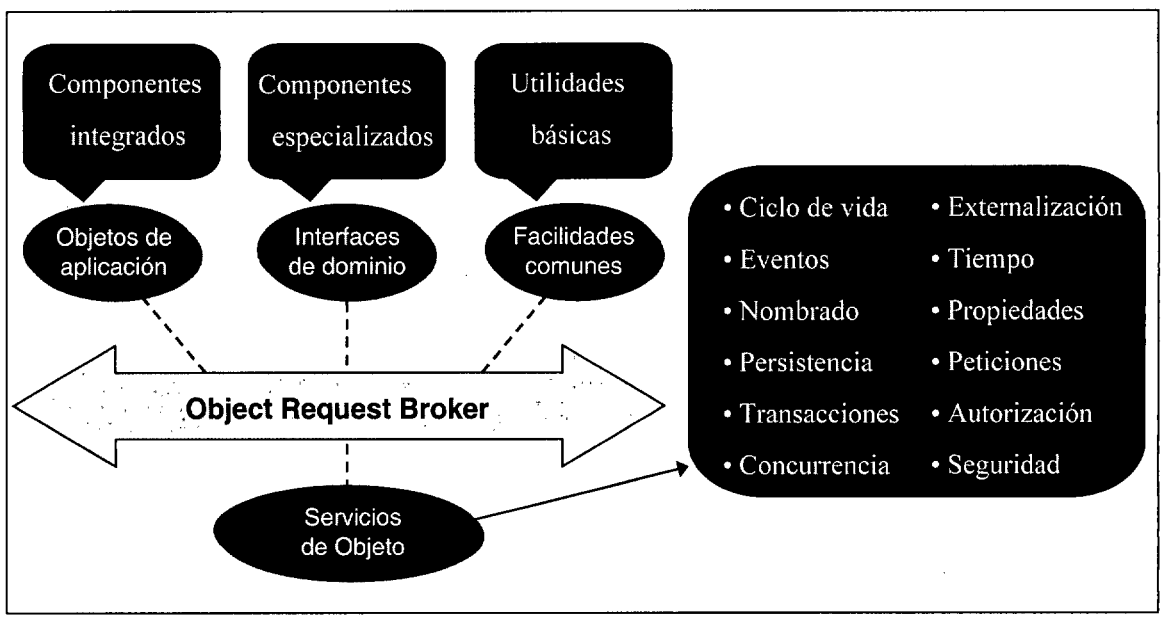


Figura 29. Arquitectura de gestión de objetos definida por el OMG

En el mismo año 1989, el OMG presentaba la arquitectura de peticiones basada en un bus de objetos (***Common Object Request Broker Architecture***, CORBA) [CORBA], con el objetivo de implementar el bus de objetos de la arquitectura OMA y de individualizar clientes y objetos servidores de tal forma que fuese posible implementarlos por separado en cualquier lenguaje, bajo cualquier sistema operativo, en cualquier plataforma hardware y ubicándolos en cualquier nodo de la red. Esto permitiría que los programadores utilizasen los objetos remotos como si fuesen locales y como si estuviesen escritos en el mismo lenguaje que los clientes. Para todo ello, CORBA ofrece a los implementadores de aplicaciones el soporte de unos determinados servicios de objeto que facilitan toda esta transparencia (en general, las versiones comerciales de CORBA no soportan todos los servicios definidos en la especificación).

CORBA sin embargo, no alcanzó verdadera popularidad hasta la versión 1.1 que surgió en el año 1992, cuando apareció el lenguaje IDL para la definición de interfaces de los objetos. En el año 1996 se hace pública la versión 2.0 cuya principal aportación es la aparición del protocolo GIOP, que generaliza la arquitectura permitiendo la conexión de diferentes ORBs. La última versión (2.6, de Diciembre de 2001), añade un capítulo de interoperabilidad segura a las funcionalidades especificadas ya en versiones anteriores: tolerancia a fallos, tiempo real, mensajería, interoperabilidad con sistemas que no sean CORBA, puentes inter-ORB, interrelación COM/CORBA, etc.

Actualmente hay abierto un proceso de especificación iniciado en el año 1999 y cuya finalización se prevé para el año 2002, de lo que se ha dado en llamar CORBA 3.0 y que especifica un modelo de calidad de servicio y de componentes para la arquitectura estándar.

Es importante reseñar que en línea con el funcionamiento habitual del OMG, lo que CORBA define no es sino el conjunto de convenciones y reglas en forma de protocolos y servicios para comunicaciones entre procesos que deben seguir todos los implementadores de la tecnología CORBA (Figura 29).

3.4.2 Detalles de CORBA

La idea principal en esta plataforma, es que cualquier implementación de CORBA que se corresponda con las interfaces especificadas y se adhiera a los protocolos definidos será capaz de comunicarse con cualquier otra implementación de CORBA. Así, los objetos pueden invocar métodos sus respectivos métodos sin tener en cuenta las plataformas en las que se ejecutan o el lenguaje en el que fueron escritos (hay implementaciones CORBA para Unix, Windows, Linux, AS/400, etc. y para lenguajes como C, C++, Java, Ada, LISP, Python o Cobol).

Siguiendo el esquema middleware común definido anteriormente (Figura 28), la Figura 30 muestra la interpretación específica que se hace de dicho esquema en el caso de CORBA.

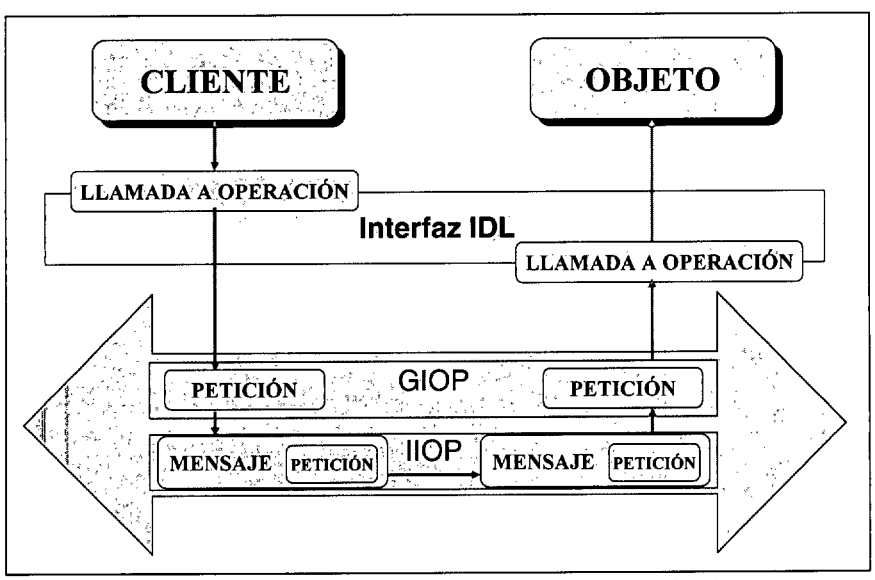


Figura 30. Modelo de invocación remota CORBA

Un cliente solamente puede invocar métodos que estén especificados en la interfaz CORBA del objeto servidor y que debe haber sido definida utilizando el lenguaje de definición de interfaces de OMG (IDL). Un compilador de IDL será el encargado de generar en el lenguaje de programación correspondiente, los cabos (*stubs*) para permitir a las implementaciones clientes la invocación de los métodos remotos definidos en el IDL del servidor y los esqueletos (*skeleton*), para asociar las implementaciones del objeto servidor al ORB en el que se vaya a ejecutar.

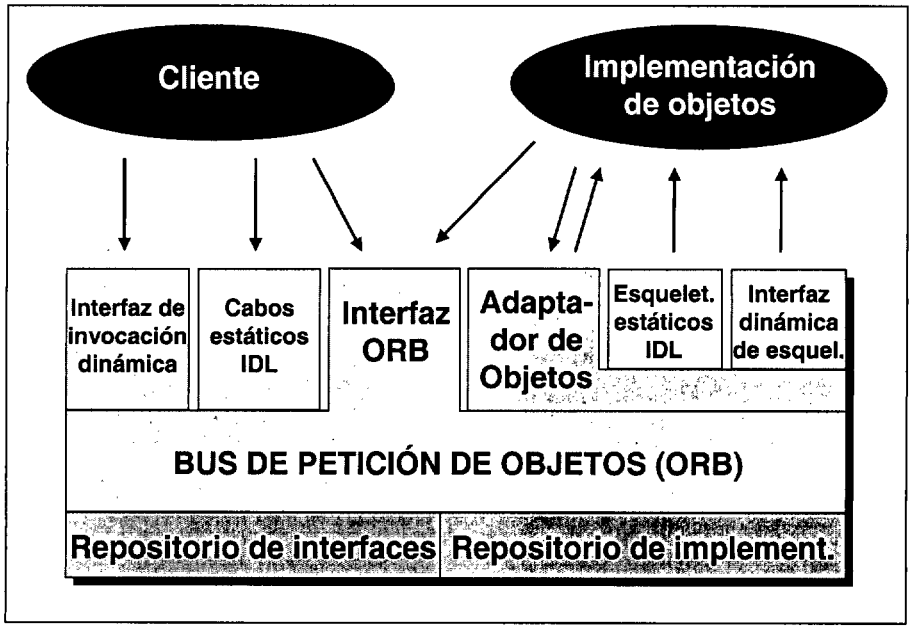


Figura 31. Visión conceptual de un ORB

Así pues, el cliente invoca a través de la correspondiente interfaz local IDL (obtenida bien dinámicamente o mediante el acceso a un repositorio de interfaces) la operación que estime oportuna en el objeto remoto.

Después de cada invocación, las subrutinas de la interfaz llamarán al ORB para transformar esas invocaciones, asociadas a un lenguaje y plataforma concretos, en una petición genérica GIOP (*Generic Inter-ORB Protocol*). GIOP define un formato de representación de datos común (basado en el formato de datos neutro CDR, *Common Data Representation*) para todos los tipos de datos de IDL así como siete formatos de peticiones básicos que cubren todas las posibles interacciones entre clientes e implementaciones de objetos.

Posteriormente se pasa la petición GIOP a la correspondiente capa de transporte de datos (en el caso de TCP/IP, dicha capa es IIOP, *Internet Inter-ORB Protocol*, que es el protocolo que se definió para hacer el transporte del protocolo GIOP genérico sobre mensajes TCP/IP de cara a resolver los problemas de comunicación planteados en Internet). Dicha capa se encarga de transformar la información de ubicación de objetos que tiene la petición GIOP en direcciones específicas del protocolo de transporte que se utilice (dirección IP y puerto TCP).

El ORB del cliente envía el mensaje al ORB donde reside la implementación del objeto (en el caso de que el objeto remoto esté en el mismo ORB que el cliente, el ORB se encarga directamente de localizar y ejecutar la operación correctamente).

La capa IIOP del ORB remoto se encarga de extraer la petición GIOP y entregarla a la capa que maneja dichos mensajes. Esta capa transforma la petición genérica en otra petición específica que dependerá del lenguaje de implementación y la plataforma de ejecución del objeto.

Finalmente las subrutinas generadas por el compilador IDL a partir de la interfaz del objeto, invocarán el método correspondiente de la implementación del objeto. Acabada la

ejecución las subrutinas de la interfaz del objeto devuelven el resultado a las de la interfaz cliente que finalmente lo entregan a la aplicación cliente.

3.4.3 Interacción CORBA-Java

Como estándar tecnológico más importante de computación heterogénea, CORBA es capaz de proporcionar al lenguaje Java un marco distribución de objetos, diferentes servicios que dicho marco soporta y la interoperabilidad de Java con otros lenguajes de programación.

De la misma forma que CORBA proporciona transparencia de distribución, Java le puede proporcionar a CORBA transparencia en cuanto a implementación, gracias a las características de portabilidad de presenta.

A partir de la versión 1.2 de Java (Java 2), Sun decidió incorporar a la plataforma de desarrollo toda la tecnología CORBA necesaria como para que las aplicaciones Java pudiesen utilizarla sin depender de implementaciones externas.

De esta forma, la plataforma Java incorpora una implementación del ORB de CORBA, APIs para el modelo de programación IDL y APIs para el modelo de programación RMI. Se comentará a continuación el primero de estos modelos, dejando para el apartado 3.5.7 el segundo (una vez se haya explicado RMI).

3.4.3.1 Java Interface Definition Language

El modelo de programación IDL, o directamente **Java IDL** [JIDL], incluye como parte integrante del núcleo de la plataforma Java 2, una implementación en Java de un ORB y el compilador `idlj` que permite compilar la definición de las interfaces hechas en IDL a Java (generando cabos, esqueletos, etc.). De esta forma puede utilizarse la plataforma Java para definir, implementar y acceder a objetos CORBA.

Recientemente RMI-IIOP (ver sección 3.5.7) ha dejado obsoleto el Java IDL en la mayoría de las aplicaciones, puesto que permite que los desarrolladores escriban interfaces compatibles con CORBA directamente en Java en vez de tener que usar el lenguaje IDL específico de OMG que hay que usar en Java IDL.

Por eso, más que profundizar en las particularidades de Java IDL, desde punto de vista de la distribución que aporta Java a este nivel, se examinarán detalladamente las características tanto de RMI como de RMI-IIOP.

3.5 Remote Method Invocation

3.5.1 Introducción

La invocación remota de métodos RMI (***Remote Method Invocation***) [Sun99] y [RMI] es un modelo de computación distribuida de objetos desarrollado por *Sun Microsystems* (forma parte del núcleo de las JDK desde la versión 1.1). A diferencia de CORBA, el único lenguaje y plataforma sobre el que se ejecuta RMI es sobre Java.

RMI posibilita que el programador cree aplicaciones distribuidas (Java-Java) de tal manera que los métodos de objetos remotos Java se puedan invocar desde otra máquina virtual que se esté ejecutando en el mismo o en otro ordenador (con el mismo o diferente sistema operativo). Como ocurría en las tecnologías que se han presentado previamente, la invocación remota de un método tiene la misma sintaxis que la invocación local de objetos (es transparente tanto para el usuario como para el programador).

RMI no requiere un mapeo de IDL a Java, ya que su arquitectura está basada únicamente en interfaces Java y puesto que RMI es una solución totalmente orientada a Java, no hay necesidad de utilizar un ORB que salve la heterogeneidad del lenguaje.

Además de esto, el proceso de desarrollo de la aplicación y el mecanismo de comunicación son, como se explicará posteriormente, casi el mismo que el procedimiento genérico descrito en la sección 3.3.

3.5.2 Mecanismo RMI

En RMI, cualquier objeto cuyos métodos se quieran hacer accesibles para otra máquina virtual se denomina objeto remoto. Dichos métodos pueden ser invocados por cualquier cliente remoto que resida en el mismo espacio de direcciones o en máquinas diferentes conectadas a Internet.

Para permitir que un objeto sea accesible de manera remota, lo primero que hay que hacer es declarar en la correspondiente interfaz remota los métodos que se quieren proporcionar (ver Figura 28) así como añadir una implementación para cada uno de esos métodos y para la interfaz `java.rmi.Remote`. Como ya se ha dicho, esta definición se escribe totalmente en Java y a partir de ella el compilador de RMI (`rmic`) generará los cabos y esqueletos correspondientes.

Cuando el objeto a invocar es de los que se ha denominado *remotos*, el cliente RMI necesita un proxy local sobre el que invocar los métodos. Dicho proxy es lo que se ha llamado cabo (*stub*) y es el encargado de recibir la invocación local, serializar todos los datos relacionados con la llamada (convertirlos en una secuencia de bytes) y enviar los datos al objeto remoto haciendo uso del protocolo de transporte correspondiente (típicamente JRMP, ver 3.5.5 o IIOP, ver [CORBA]). Los datos para el objeto remoto los

recoge el esqueleto (*skeleton*), que se encarga de hacerle a los datos el procesado inverso al que les hace el cabo y de entregarlos a la implementación real del objeto para que los procese. Después serializa la respuesta y la envía de vuelta al cliente. En la plataforma Java 2, el propio entorno de ejecución contiene código genérico capaz de realizar la tarea de los esqueletos, por lo que la generación de los mismos es opcional (no así la de los cabos, que es obligatoria para que el cliente siga pudiendo hacer la invocación remota).

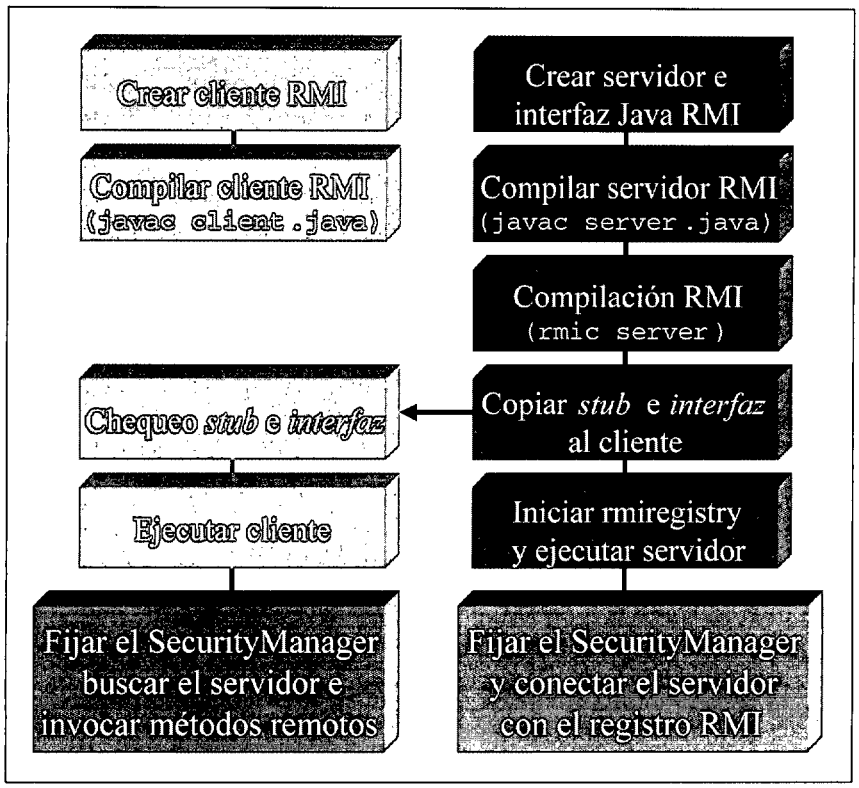


Figura 32. Definición de una aplicación RMI

Para que un cliente se comunique con un objeto que se haya ubicado en un determinado servidor (realmente con quien se comunica es con la 'interfaz remota del objeto'), es necesario que lo encuentre primero (que obtenga una referencia de dicho objeto). Para ello, todo objeto remoto debe registrarse en el RMIRegistry, que no es sino un proceso que escucha en un puerto bien conocido (por defecto 1099) y que gestiona una tabla donde se mantienen emparejados objetos con nombres únicos. Cuando un cliente quiere acceder a un objeto, sólo tiene que proporcionar su nombre (con una RMI-URL del estilo de `rmi://www.it.uc3m.es/nombre_de_objeto`). El RMIRegistry localiza el esqueleto del objeto que se ha pedido, crea un nuevo objeto que se ubica en memoria y se le proporciona el cabo del objeto al cliente.

Los cabos locales ofrecen los mismos métodos que el objeto remoto, por lo que usando la referencia remota el cliente puede invocar todos estos métodos como si estuviesen ubicados en su máquina virtual local.

Adquirida la referencia de un objeto remoto, el cliente empieza a hacer *pings* al servidor donde reside dicho objeto, con el fin de mantener abierta la línea de conectividad. Pasado un tiempo sin recibir *pings* del cliente, el servidor elimina los objetos remotos correspondientes.

De cara a conseguir simetría en el modelo de comunicación se definen los llamados *callbacks* (también existen en CORBA), que permiten que un cliente pueda ser configurado como un servidor y recibir invocaciones del servidor, que ahora haría el papel de cliente.

3.5.3 Prestaciones de RMI

3.5.3.1 Serialización de objetos

En Java, los tipos primitivos se pasan por copia (por valor) y los arrays y objetos por referencia.

Cuando se invoca un método RMI, todos los parámetros se pasan por valor (por copia), es decir que los parámetros se copian de una máquina a otra cuando se hace la invocación del método (los cambios efectuados en estos parámetros copiados, evidentemente no tienen repercusión en los parámetros originales que residen en el objeto que hace la invocación). El problema de este esquema, es que en RMI, además de tipos primitivos de Java, también se pueden pasar como parámetros objetos enteros y si un objeto tiene referencias a otros objetos, en el espacio de direcciones remoto dicha referencia puede no tener sentido.

Para resolver este problema se utiliza la serialización de forma generalizada: se convierte el objeto Java y todos los objetos que incluya como atributos (salvo que estén específicamente definidos como `transient`) en una ristra de bits que puede ser directamente enviada por la red, almacenarse en disco, etc. Por ejemplo, si se serializa un `Vector` que contenga objetos, cuando se des-serialice, se obtendrá dicho `Vector` con todos los objetos dentro (es un mecanismo sencillo para almacenar cosas en disco, pues basta una simple invocación para guardar y otra para recuperar). Esto sin embargo, genera a su vez otro problema y es que si el objeto que se quiere pasar es muy grande (se incluyen por ejemplo, referencias a muchos otros objetos) el mecanismo puede ser altamente ineficiente (es posible no obstante, que el usuario se programe a medida los métodos de serialización, `writeObject` y `readObject`, para aplicarle un mecanismo de compresión a los datos antes de ser transmitidos, por ejemplo).

Puede probarse fácilmente lo que realmente se transmite, sin más que escribir a un fichero el resultado de la serialización:

```
import java.io.*;

...
        try {
                FileOutputStream ostream = new FileOutputStream("serial.tmp");
                ObjectOutputStream p = new ObjectOutputStream(ostream);
                p.writeObject(solution);
                p.flush();
                ostream.close();
        } catch(Exception ex) {
                System.out.println("Falló la serialización:"+ex.toString());
        }
...
```

Tabla 6. Código para comprobar la serialización RMI

RMI es capaz también de simular el paso por referencia (en vez de copiar el objeto, se manda una referencia a éste con lo que el método remoto trabajaría con el objeto original). Para ello, el parámetro a pasar debe ser obligatoriamente a su vez un objeto remoto (de otra forma se pasaría siempre por valor) y lo que se acaba enviando al pasar el parámetro, es su cabo (que es serializable y por lo tanto puede ser convertido en una ristra de bits) con lo que las operaciones sobre ese parámetro son realmente invocaciones remotas (que se ejecutarán en el ordenador local en vez de en el ordenador remoto).

En resumen, se puede decir que todos los tipos primitivos de Java se pasan por valor y se serializan automáticamente (los cambios en los datos en remoto no afectan a los datos locales). Para pasar un objeto entero por valor, debe ser serializable (suele bastar con implementar la interfaz `java.io.Serializable`, que es simplemente una interfaz vacía que permite controlar qué objetos se pueden pasar y cuáles no) y así se hace una copia exacta de todo el objeto (datos y grafo de objetos que pueda contener salvo que estén definidos como `transient`) en la máquina virtual remota. Si se quiere pasar por referencia, tiene que ser un objeto remoto con sus correspondientes interfaces RMI (eso exige que el servidor cree un objeto esqueleto y otro cabo para devolver al cliente, lo cual puede resultar también bastante costoso).

Serializar un tipo primitivo no cuesta prácticamente nada. Objetos simples del estilo de un '`String`' cuestan algo más, objetos remotos (cabos) más aún y por último objetos con objetos incluidos dentro todavía mucho más.

3.5.4 Modelo de invocación

En las invocaciones RMI, siempre se asume que el objeto es remoto (tiene interfaz remota y está ubicado en otra máquina). Es decir que incluso en el caso de que los objetos estén en la misma máquina virtual, si la referencia de una objeto remoto se obtiene haciendo uso del RMIRegistry (se obtiene el cabo, etc.) el procedimiento RMI se ejecuta entero desde la serialización, a la llamada TCP (otra cosa es que la referencia de los objetos remotos se obtenga de instanciarlos como objetos normales, lo cual se podría hacer si están ejecutándose en la misma máquina y siempre que no se esté en un entorno de EJBs, ver [J2EE].

Es evidente por otro lado, que una invocación remota exige mucho más trabajo que la invocación local, pues el retardo total que percibirá el cliente será el resultado de: serializar la llamada, latencia de la transmisión, des-serializar, procesamiento de la invocación, serializar la respuesta, nueva latencia de transmisión, des-serializar y procesamiento final de la respuesta. Además, cada vez que se hace una invocación remota, el servidor se ve obligado a crear dos objetos nuevos (cabo y esqueleto) con el correspondiente gasto de memoria.

Por todo ello, a la hora de diseñar una aplicación con invocaciones remotas, es preciso estudiar en detalle en qué van a consistir exactamente estas invocaciones. En ocasiones se pierde más tiempo en la fase de serialización que lo que se tarda en ejecutar el método correspondiente, con lo que un acceso RMI en local puede no tener sentido. Hay que estudiar si es mejor pasar un objeto complejo, o bien pasar la referencia de un objeto remoto y hacer múltiples llamadas remotas para obtener sus atributos o hacer una

invocación remota que coja muchos parámetros de una vez o muchas invocaciones que cojan atributo a atributo, etc.

3.5.5 JRMP

JRMP (*Java Remote Method Protocol*) es el protocolo que utiliza la tecnología RMI para transportar los datos necesarios para llevar a cabo la invocación remota (parámetros de llamada, métodos, excepciones, etc.). Su funcionamiento está basado en otros dos protocolos: el protocolo de serialización de objetos (del cual ya se ha comentado algo en apartados anteriores), utilizado para convertir los datos en una ristra de bitss para enviarlos y recibirlos por la red y HTTP cuyo método 'post' se utiliza en determinadas circunstancias para encapsular llamadas RMI y hacerlas pasar a través de un firewall.

JRMP funciona normalmente sobre TCP en un puerto que se asigna dinámicamente (no es un puerto de los denominados bien-conocidos y de ahí el posible problema de seguridad con los firewalls) y tiene definidos dos tipos de mensajes: de entrada (*in*) y de salida (*out*), siempre desde el punto de vista del cliente.

OUT

Los mensajes de salida están compuestos por una cabecera y un cuerpo (este último es el que se serializa y en ocasiones se encapsula en un mensaje HTTP).

- Cabecera: `0x4a 0x52 0x4d 0x49` + *Versión* (`0x00` ó `0x01`) + *Protocolo* (`0x4b`, `0x4c` ó `0x4d`) + *Identificador* (`IP + puerto`)
- Cuerpo: Call (`0x50 + datos`), Ping (`0x52`), DgcAck (`0x54 + identificador`)

IN

Los mensajes de entrada pueden ser de tres tipos:

- ReturnData: Protocol ACK (0x4e) + Returns (0x51 + valor)
- Ping ACK: Protocol ACK (0x4e) + Ping ACK (0x53)
- HttpReturn

Para enviar los datos, la capa de transporte RMI intenta abrir sockets directamente con los ordenadores destino. En el caso de que el puerto 1099 del RMIRegistry esté restringido por un firewall, RMI prevé un mecanismo para encapsular sus mensajes sobre HTTP. Lo que se hace es simular que el RMIRegistry está escuchando en el puerto 80, para lo cual se encapsulan los datos RMI en peticiones HTTP (HTTP tunneling). El servidor Web que debe residir en la misma máquina que el RMIRegistry coge la invocación y mediante un CGI que proporciona Sun (en la plataforma Java 2), redirige la invocación hacia el puerto donde escucha el RMIRegistry. Finalmente las respuestas se encapsulan en el cuerpo de un response de HTTP. Todo este proceso lo hace de forma transparente y automática la clase `RMISocketFactory` y los sockets servidores (el primero intenta la conexión HTTP si falla la conexión directa, los segundos entienden mensajes en el puerto 80). Este mecanismo evidentemente es más lento que el transporte habitual y además restringe la actuación del servidor que no puede invocar métodos en el cliente (*callbacks*).

3.5.6 Ejemplo RMI

Al igual que se ha visto en el apartado relativo a HTTP, se van a analizar en detalle los diferentes mensajes que se originan en una red TCP/IP al realizar una invocación remota de un método sobre un objeto, utilizando el mecanismo RMI.

El caso que se evalúa en el ejemplo es un applet que se carga en el navegador del cliente a partir de una página Web. Dicho applet únicamente incluye un botón cuya pulsación permite incrementar un contador que está en el objeto servidor (Figura 33).

La secuencia de mensajes que tiene lugar hasta el momento en que el applet está cargado en el cliente es muy similar a la que se vio en el ejemplo de HTTP (ver Figura 13):

- Se obtiene la página HTML en la que se encuentra la referencia al applet que hay que descargar (fichero `rmi.html` en este ejemplo).
- Para ello se abre una nueva conexión TCP con el servidor Web y se le pide la clase del applet (`rmiApplet.class`) mediante HTTP, antes de mostrar nada en el navegador.

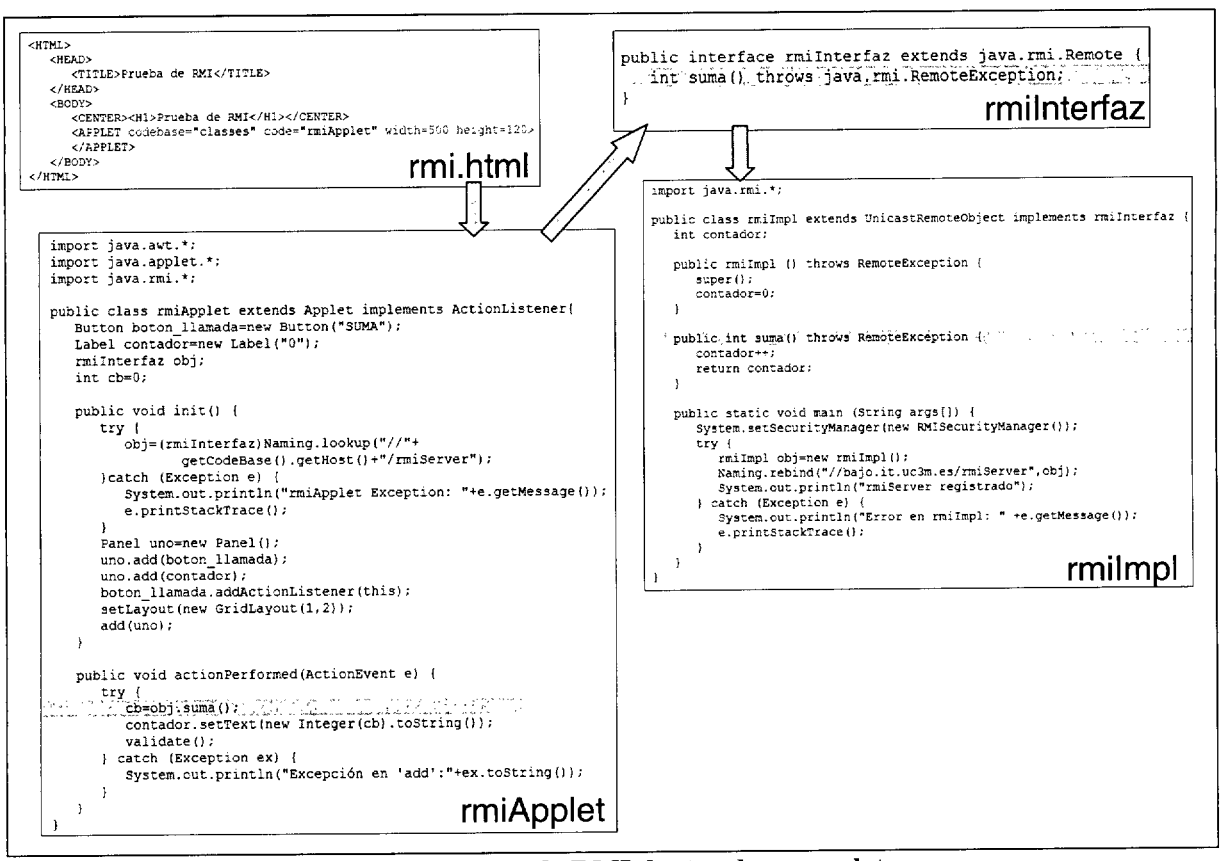


Figura 33. Ejemplo de RMI dentro de un applet

- Durante la ejecución del applet, el cargador de clases de la máquina virtual exige una nueva petición HTTP para obtener la clase de la interfaz del objeto remoto (`rmiInterfaz.class`).

- Posteriormente el cliente inicia una nueva conexión TCP con el puerto 1099, para obtener del `rmiregistry` el nombre de la clase del cabo del objeto remoto y el número de puerto en el que escuchará el objeto remoto (puerto JRMP asignado dinámicamente).
- El cliente hace una petición HTTP (sobre la primera conexión TCP abierta) para obtener el fichero `rmiImpl_stub.class`.
- Tan pronto como se pulsa el botón en el applet cliente, se abre una nueva conexión TCP con el puerto que se le había comunicado en el mensaje anterior y tras una serie de mensajes de configuración de la sesión, se realiza la invocación remota sobre JRMP.

3.5.7 RMI sobre IIOP

RMI permite escribir aplicaciones distribuidas escritas en Java sin el requisito de la utilización de un lenguaje separado para la definición de interfaces y utilizando JRMP para la comunicación entre objetos remotos Java. El principal problema de RMI es la total carencia de interoperabilidad con otros lenguajes y por supuesto con cualquier objeto CORBA (aun cuando dicho objeto esté también implementado en Java) puesto que el protocolo JRMP no es estándar.

Por otro lado, IIOP es el protocolo de comunicación estándar de CORBA que, como ya se ha dicho, define el mecanismo de envío de bits por el medio de transmisión entre un cliente y un servidor CORBA cuyas interfaces hayan sido declaradas utilizando IDL. Tal y como se comentaba en la sección 3.4.3.1, la implementación que las JDK hacen de la especificación CORBA/IDL es conocida como Java IDL y permite que objetos CORBA sean accedidos desde el lenguaje de programación Java.

La implementación de la referencia de RMI sobre IIOP [Sun02c], fue desarrollada de forma conjunta por Sun e IBM (Junio de 1999) y se ejecuta sobre JDK 1.1 (desde la versión 1.1.6 en adelante) y por supuesto en Java 2. La idea es ser capaz de combinar el sencillo mecanismo de uso de RMI (con algunas restricciones) y el protocolo de interoperabilidad de CORBA para permitir que objetos RMI se puedan comunicar con ORBs que soporten la especificación CORBA 2.3 (versiones más antiguas de CORBA no soportan el estándar del OMG de paso de objetos por valor en el que se basa RMI-IIOP).

Con RMI sobre IIOP, los desarrolladores pueden escribir las interfaces remotas en el lenguaje de programación Java e implementarlos usando la tecnología Java y las APIs de Java RMI. Estas interfaces pueden ser implementadas en cualquier otro lenguaje (que tenga el correspondiente mapeo definido por el OMG y el correspondiente ORB disponible) utilizando el IDL derivado de las interfaces remotas definidas en Java. De la misma forma, los clientes pueden ser implementados en otros lenguajes. Utilizando RMI-IIOP, los objetos pueden pasarse tanto por valor como por referencia sobre IIOP.

Aunque anteriormente no era parte del núcleo de las JDK, ahora sí que se incluye en la distribución la versión actualizada del compilador `rmic` de RMI, cuyo modificador `-iiop`, permite la generación de cabos IIOP y la generación automática de código IDL. Esto último es un punto muy importante de cara a conseguir la interoperabilidad RMI-CORBA.

Un cliente RMI-IIOP por definición no tiene por qué tener la capacidad de acceder a cualquier objeto CORBA existente. La semántica de los objetos CORBA definidos en IDL engloba a la de aquellos objetos definidos para RMI-IIOP por lo que el IDL de objetos CORBA ya existentes no siempre pueden ser traducidos a una interfaz Java de RMI-IIOP. Solamente cuando la semántica específica de un objeto CORBA se corresponda con la de RMI-IIOP, será posible que un cliente RMI-IIOP invoque un objeto CORBA existente.

Al final, la diferencia entre un objeto CORBA y uno RMI-IIOP es únicamente una cuestión de implementación así que esta incompatibilidad puede ser relativamente resuelta. Cuando se diseña un nuevo objeto distribuido con una interfaz Java para RMI-IIOP, es posible que automáticamente se genere su correspondiente IDL con la utilidad `rmic` (y también sus correspondientes cabos IIOP para posibles implementaciones Java). A partir de este fichero IDL puede ser implementado el objeto CORBA (en C++ por ejemplo). Este objeto es totalmente operativo en CORBA por lo que puede ser accedido por cualquier cliente CORBA y además puede ser accedido por un cliente RMI-IIOP sin ningún tipo de limitaciones. Para éste último, este objeto CORBA C++ aparece como un objeto RMI-IIOP puro, puesto que el origen de su interfaz es una interfaz Java RMI-IIOP. De la misma forma, si el objeto se implementa en Java para ser accedido mediante RMI-IIOP, también aparecerá como un objeto puro de CORBA, puesto que un cliente CORBA puede acceder a él mediante su interfaz IDL.

Si el objetivo final es utilizar Java para acceder a objetos CORBA que ya han sido escritos, Java IDL es la alternativa a RMI-IIOP. Usando Java IDL, es posible acceder a cualquier objeto CORBA desde Java. Y si por el contrario el objetivo es exportar la funcionalidad de un objeto Java RMI a usuarios CORBA, RMI-IIOP es lo que debería usarse.

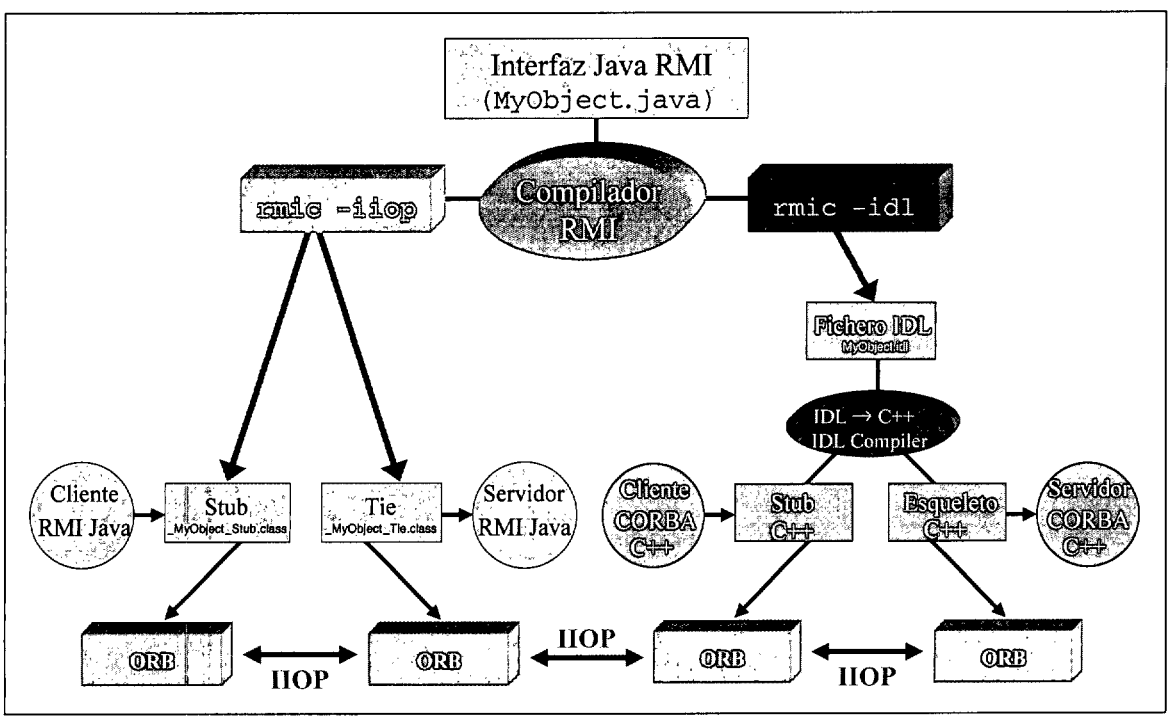


Figura 34. Compatibilidad CORBA / RMI

Puesto que RMI-IIOP soporta los protocolos JRMP e IIOP, ha habido mucha especulación sobre el hecho de que RMI (JRMP) debería ser integrado en RMI-IIOP (este

último por cierto tiene mayor sobrecarga que el primero). Sin embargo, Sun ha anunciado que de momento es su intención seguir soportando los tres mecanismos: Java IDL, RMI (JRMP) y RMI-IIOP.

3.6 Distributed Computing Environment

El entorno de computación distribuida DCE (***Distributed Computing Environment***) [DCE], es un conjunto de tecnologías estándares independientes de cualquier fabricante desarrollado por el *Open Software Foundation* (ahora *Open Group*).

Como ocurre en otras alternativas, DCE utiliza un lenguaje de definición de interfaces basado en IDL y C para especificar qué métodos puede ser accedidos de manera remota por los demás objetos. También aquí existe la idea de cabos, que son llamados por la aplicación cliente a modo de proxy para invocar un servicio y esqueletos de servidor que reciben estas invocaciones del cliente e invocan en la implementación real del correspondiente servicio.

En DCE, los sistemas distribuidos se organizan en celdas, que son conjuntos de recursos, servicios y usuarios que tienen funcionalidades comunes y comparten un conjunto común de servicios DCE. Algunos de estos servicios y tecnologías que pueden usarse dentro de las celdas son: servicio de directorios, servicio de distribución de ficheros, distribución de tiempo, invocaciones remotas o diversos hilos de ejecución. Incorpora además un completo servicio de seguridad basado en el Kerberos del MIT.

Uno de los problemas más importantes de DCE hoy en día es que el DCE está basado en RPC y por lo tanto está orientado a procedimiento. Aunque es capaz de soportar un modelo neutral de objetos, originalmente se diseñó para programación procedimental y de hecho no acaba de adaptarse bien a aplicaciones distribuidas diseñadas con orientación a objetos. Soporta C y C++ aunque hay también pasarelas a otros lenguajes (como Cobol o Ada) disponibles.

3.7 Distributed Component Object Model

El modelo de componentes de objetos distribuidos (***Distributed Component Object Model*** DCOM) [DCOM], es un protocolo de nivel de aplicación para invocaciones remotas orientadas a objeto y de hecho también es denominado 'RPC de objetos' u ORPC. El protocolo consiste en una serie de extensiones que son capaces de interactuar con los servicios de COM (*Component Object Model*) y está basada en la especificación del entorno de computación distribuido (DCE).

COM es una tecnología de Windows basada en OLE (*Object Link Embedding*) que permite que objetos Windows de carácter general interactúen y puedan ofrecerse servicios entre ellos. Con un lenguaje de definición de interfaces muy parecido al de

DCE, los objetos pueden publicar sus métodos accesibles y utilizando las librerías de COM, invocarse entre ellos sin depender del lenguaje de implementación.

DCOM (anteriormente llamado *Network OLE*) es básicamente COM distribuido, de tal manera que permite a los objetos COM acceder transparentemente a otros objetos COM remotos con la misma filosofía que en entornos anteriores: definición de interfaces (utilizando *Microsoft's Interface Definition Language*, MIDL), generación de cabos, registro de objetos, etc.

DCOM también soporta seguridad en aplicaciones, integrando y actualizando el modelo de seguridad de Windows NT para llevar a cabo autenticación, cifrado de mensajes, etc.

Aunque DCOM es un producto Microsoft es un estándar abierto que ha sido portado a otras plataformas como Unix.

3.8 Simple Object Access Protocol

3.8.1 Introducción

El protocolo simple de acceso a objetos SOAP (***Simple Object Access Protocol***) [SOAP] y es un nuevo protocolo no estándar muy ligero desarrollado por el World Wide Web Consortium (W3C) para intercambios de información en un entorno descentralizado.

Uno de sus principales objetivos es la reutilización de la infraestructura existente en Internet para permitir a las aplicaciones la comunicación directa unas con otras sin necesidad de tener que tomar medidas especiales para evitar ser bloqueados por cortafuegos [Cha99]. En general los cortafuegos se configuran para permitir tráfico en el puerto 80 (puerto HTTP bien conocido) y para bloquear la mayoría de los otros puertos, por lo que casi todos los protocolos clásicos de distribución son bloqueados por defecto si no se hacen cambios en la arquitectura de comunicaciones y algunos incluso utilizan una asignación dinámica de puertos, por lo que el problema es incluso peor).

La principal idea que hay detrás de SOAP, es la encapsulación de un protocolo de distribución de objetos encima de HTTP para poder utilizar sus primitivas básicas (GET, PUT, POST, etc.) y así posibilitar el intercambio de datos entre el navegador y el servidor Web.

La especificación describe cómo los mensajes SOAP pueden ser transportados por HTTP ofreciendo un mecanismo sencillo para intercambiar información estructurada usando XML. Además, desde la última versión (1.2 del año 2001) también se soporta la invocación remota de métodos utilizando XML para definir el formato de las peticiones y las respuestas y usando el habitual comando HTTP POST para enviar esta información.

SOAP consta de tres partes:

- La estructura ***envoltorio*** de SOAP define el marco general para expresar lo que contiene un mensaje, qué entidades tratarán con él y si es opcional u obligatorio. Es el elemento más externo del documento XML y representa el mensaje SOAP junto con una cabecera opcional y el cuerpo del mensaje SOAP.
- Las ***reglas de codificación*** SOAP definen en XML el mecanismo de serialización que puede ser usado para intercambiar instancias de los tipos de datos definidos por las aplicaciones. Estos tipos, son los clásicos tipos disponibles en todo lenguaje de programación (tipos simples como enteros, reales o cadenas y compuestos como estructuras o matrices).
- La ***representación RPC*** de SOAP define una convención que puede usarse para representar invocaciones remotas a objetos y respuestas. El hecho de utilizar SOAP para RPC es independiente del funcionamiento básico del protocolo de transporte que haya por debajo. En el caso de utilizar HTTP como protocolo básico de transporte, una llamada tipo RPC se mapea de forma natural en una petición HTTP y la respuesta en una respuesta HTTP. Sin embargo, la utilización de SOAP para RPC no está limitada al protocolo HTTP (puede usarse también FTP o SMTP, por ejemplo). El receptor del mensaje (típicamente un servlet) es el responsable de entregar la invocación al objeto correspondiente.

3.8.2 Ejemplo

Para poder entender mejor cómo funciona SOAP, sería de gran utilidad ver cómo se lleva a cabo la ejecución remota de un método.

En el ejemplo, se supone que el usuario ha contactado con una determinada tienda con el objetivo de comprar un libro. La sintaxis genérica del método que hay que ejecutar para ordenar la compra de un libro es la siguiente:

```
boolean OrderBook (in Title string, in Author string, out Price integer)
```

Cuando la aplicación cliente invoca este método y después de aplicar el mapeo correspondiente la petición tendría el aspecto de un HTTP POST estándar, en el que el cuerpo del mensaje está expresado en XML (los atributos del método `Title` y `Autor` se pasan como etiquetas XML):

```
POST /ShoppingServer HTTP/1.1
Host: www.choppingcenter.com
Content-type: text/xml
Content-Length: nnnn
SOAPMethodName: Some-Namespace-URI#OrderBook

<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
      <SOAP:Body>
            <m:OrderBook xmlns:m="Some-Namespace-URI">
                  <Title>Alice in wonderland</Title>
                  <Author>Lewis Carrol</Author>
            </m:OrderBook>
      </SOAP:Body>
</SOAP:Envelope>
```

Tabla 7. Ejemplo de petición SOAP

Una vez que el mensaje se ha recibido y procesado por el servidor, el servidor puede devolver una respuesta ejecutando el método `OrderBookResponse` de la siguiente forma:

```
HTTP/1.1 200 OK
Connection: Close
Content-type: text/xml
Content-Length: nnnn

<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
      <SOAP:Body>
            <m:OrderBookResponse xmlns:m="Some-Namespace-URI">
                  <return>1</return>
                  <Price>10</Price>
            </m:OrderBookResponse>
      </SOAP:Body>
</SOAP:Envelope>
```

Tabla 8. Ejemplo de respuesta SOAP

3.9 Comparativa

Dada la diversidad de tecnologías que se han presentado, se propone en este apartado a modo de síntesis de cada una de ellas, las ventajas e inconvenientes que se estiman más importantes, teniendo en consideración la repercusión que dichas tecnologías puedan tener como fundamento de las arquitecturas multicapa y en general como base para la implementación de mecanismos de comunicación distribuida de una plataforma de intermediación electrónica.

3.9.1 CORBA

3.9.1.1 Ventajas

- Puesto que CORBA fue desarrollado por el OMG (avalado actualmente por más de 800 empresas), se ha convertido en un estándar abierto para la industria y hay actualmente gran cantidad de productos que lo apoyan (IDL, IIOP, etc.).
- La especificación y las diferentes implementaciones de CORBA son bastante estables, puesto que ambas han existido y se han desarrollado desde principios de los 90.
- CORBA proporciona gran cantidad de servicios de objeto estándares que pueden ser utilizados para desarrollar aplicaciones mucho más fácilmente. Aunque depende de cada implementación el hecho de proporcionarlos o no, la especificación define gran cantidad de ellos: *colecciones*, *concurrencia*, *eventos*, *externalización*, *registro*, *ciclo de vida*, *servicio de nombres*, *notificaciones*, *persistencia*, *propiedades*, *peticiones*, *relaciones*, *seguridad*, *tiempo*, *negocio*, *transacciones* [CORBAa].
- CORBA permite comunicaciones entre objetos implementados en diferentes lenguajes porque las interfaces están escritas en un lenguaje neutral de definición (IDL) y hay gran cantidad de compiladores para generar cabos y esqueletos en el lenguaje apropiado.

- Con IDL la interfaz está claramente separada de su implementación y los desarrolladores pueden incluso crear diferentes implementaciones para la misma interfaz.
- Hay muchas implementaciones de CORBA, tanto comerciales como de libre distribución (ver [Cetus]).
- La invocación dinámica utilizando el servicio de nombres, permite que los objetos se reconozcan unos a otros de forma transparente en tiempo real.

3.9.1.2 Desventajas

- CORBA está específicamente diseñado para ser ejecutado en un entorno muy genérico y heterogéneo donde muchas plataformas y lenguajes de programación pueden ser utilizados. Si se utiliza en un entorno en el que el único lenguaje es Java, gran parte de la complejidad que introduce CORBA no es necesaria.
- Aunque ha experimentado una gran mejora, el rendimiento de CORBA todavía es cuestionable (la complejidad interna de CORBA es elevada, si se pretende utilizar CORBA en un entorno de servicios Web, obtener el ORB desde el cliente puede consumir mucho tiempo, etc.).
- CORBA no soporta transferencia de objetos o código (está limitada a la transferencia de tipos nativos y algunos tipos estructurados).
- No es capaz de llevar a cabo recolección de basura distribuida de carácter general.

3.9.1.3 Java IDL

- Es parte del núcleo del API de las JDK por lo que es gratuito y está avalado por Javasoft.
- Añade a las plataformas Java las capacidades de CORBA por lo que puede utilizarse para definir e implementar clientes en Java capaces de acceder a objetos CORBA (como ya se ha comentado antes, tampoco tiene sentido si lo único que se va a utilizar es Java).
- No implementa la mayoría de los servicios CORBA (el servicio de nombres no es persistente, no incorpora transacciones, no tiene notificación de eventos, etc.).
- Con la posibilidad de enviar mensajes RMI sobre el protocolo IIOP de OMG, la utilización de Java IDL en el entorno de los servicios Web es más que cuestionable (visto también el escaso apoyo que está teniendo CORBA en este sector, no parece muy probable que sea necesario un mecanismo del estilo de Java IDL).

3.9.2 RMI

3.9.2.1 Ventajas (ver [Rei01] y [Sun02a])

- RMI forma parte del núcleo de la plataforma Java desde la versión 1.02 de las JDK (por lo que es gratuito y viene auspiciado por Sun) y existe en todas las máquinas virtuales desarrolladas desde la versión 1.02 de la máquina virtual de Java (y por supuesto en Java 2, con notables mejoras). Todos los sistemas RMI hablan el mismo protocolo por lo que todos los sistemas Java pueden hablar entre

ellos directamente sin necesidad de sobrecargar el mecanismo con otro protocolo que actúe de puente (tipo GIOP).

- RMI hace relativamente simple la tarea de escribir servidores Java y clientes Java que puedan acceder a ellos. La interfaz remota es una interfaz Java, por lo que el mismo lenguaje se usa en la definición y en la implementación de la interfaz (es muy apropiado para aplicaciones Java-Java y desarrollos con un único modelo de objetos). RMI no se ve involucrado en los complejos mecanismos que suelen regir las plataformas *middleware* (gestión del ORB, etc.).
- RMI está orientado a objetos y puede pasar objetos completos como argumentos o valores de retorno (no se queda en los habituales tipos predefinidos básicos). Pasar objetos mediante una serialización previa a través de la red permite una utilización verdaderamente eficaz de la tecnología de orientación a objetos en un entorno distribuido como los sistemas multicapa.
- RMI utiliza los mecanismos nativos de seguridad de Java que permite que el sistema permanezca seguro cuando los usuarios cargan las correspondientes implementaciones. RMI utiliza un gestor de seguridad definido para proteger a los sistemas de applets hostiles y en general para proteger a los sistemas y las redes de código hostil que se haya podido descargar.
- RMI utiliza el mecanismo de Java de recolección de basura para eliminar los objetos remotos que no son ya referenciados por ningún cliente de la red. De manera análoga a la recolección de basura dentro de la máquina virtual, la recolección de basura distribuida permite la definición de objetos servidor según se necesiten, permitiendo que puedan ser eliminados cuando no se necesite el acceso por parte de los clientes.
- La curva de aprendizaje para desarrolladores que no tengan ninguna experiencia con RMI es mucho mejor que la de CORBA

3.9.2.2 Desventajas

- Utiliza un protocolo no estándar, JRMP, para permitir la comunicación entre objetos remotos, por lo que representa una solución de distribución parcial: no soluciona la interoperabilidad entre aplicaciones escritas en diferentes lenguajes (hay que recurrir a IIOP).
- La escalabilidad de la arquitectura RMI no ha sido completamente demostrada y quizá no sea la mejor solución para sistemas grandes con un alto grado de distribución.
- Sólo define invocaciones remotas sin proveer ningún otro servicio adicional al desarrollador de la aplicación distribuida (únicamente una versión muy ligera del servicio de nombres).
- Existe el clásico riesgo de seguridad con la ejecución remota de métodos y las limitaciones funcionales motivadas por las restricciones de seguridad.
- RMI no incluye invocación dinámica de interfaces (CORBA por ejemplo permite que los objetos se descubran en tiempo de ejecución).
- Aunque RMI está presente en Java desde las JDK 1.02 no ha sido totalmente funcional hasta la versión 1.1. Además hay muy pocos navegadores que soporten RMI directamente y casi todos requieren la instalación del plugin de Java.

3.9.2.3 RMI sobre IIOP

- Está presente desde las JDK 1.1.6, por lo que es gratis y viene avalado por Sun e IBM.
- RMI-IIOP permite escribir aplicaciones orientadas a objetos distribuidas e interoperables sin la necesidad de un lenguaje común de definición de interfaces.
- RMI-IIOP sólo puede interoperar con otros objetos CORBA cuando sus interfaces remotas se hayan definido originalmente como interfaces Java RMI (la interoperabilidad no es posible cuando originalmente se hayan definido usando CORBA IDL).
- Para utilizar RMI-IIOP en un navegador Web es necesario el plugin de Java.
- Es una tecnología bastante nueva y aún deben desarrollarse y madurarse algunos aspectos concernientes a ella: soporte para Linux, soporte para transporte seguro, etc.

3.9.3 DCE

3.9.3.1 Ventajas

- Especificación abierta adoptada por el Open Group y por el ISO. Implementación disponible por muchos fabricantes (la implementación de referencia está disponible de forma gratuita). Ha sido reimplementado a partir de especificaciones de Microsoft.
- DCE proporciona un servicio de nombres transparente a la vez que localización de objetos, autenticación, autorización, control de acceso e instanciación dinámica de servicios. También proporciona una herramienta de gestión remota.

3.9.3.2 Desventajas

- Está basado en la especificación de RPC (por lo que está orientada a proceso) No se adapta bien a aplicaciones orientadas a objeto, puesto que inicialmente se diseñó para soportar lenguajes procedimentales.
- Está diseñado para soportar C y C++ como tecnologías nativas. Puentes hacia otros lenguajes solo están disponibles a través de terceros.
- DCE tiene invocación dinámica de interfaces (no tiene repositorio de interfaces).

3.9.4 DCOM

3.9.4.1 Ventajas

- DCOM está incorporado en todos los sistemas operativos de Microsoft. Hay implementaciones para varias plataformas UNIX. Tiene el aval de Microsoft.
- Aunque se basa en el RPC de DCE, a diferencia de DCE ofrece posibilidades de programación orientada a objetos (gracias a los objetos, interfaces y métodos de COM).

- Tiene un buen soporte para la herramienta de desarrollo y proporciona un modelo completo para aplicaciones de computación distribuida.
- Proporciona interoperabilidad entre todos los lenguajes Microsoft.

3.9.4.2 Desventajas

- COM y DCOM no están basados en estándares (aunque hay implementaciones de DCOM basadas en la especificación del RPC de DCE ofrecida por el Open Group). Es una solución muy específica de Microsoft.
- DCOM no proporciona código portable ni independencia de la plataforma.
- No está claro lo efectivos que COM y DCOM pueden llegar a ser trabajando de forma aislada de por ejemplo MTS.
- Como se ha dicho antes ofrecen cierta orientación a objetos (no se diseñaron originalmente con tal idea) y mediante Microsoft Visual J++ ofrece (aunque Microsoft pretende no seguir dando soporte a Java en sus herramientas de desarrollo visuales) acceso a objetos COM y DCOM desde Java. Sin embargo, es más un puente (parche) para adaptarse a la tecnología que una extensión real del modelo de objetos de Java para soportar aplicaciones distribuidas.

3.9.5 SOAP

3.9.5.1 Ventajas (ver [SOAP])

- La especificación es accesible de forma gratuita y viene avalada por el World Wide Web Consortium.
- Está principalmente basado en estándares como XML y HTTP y puede interactuar perfectamente con los navegadores Web actuales.
- SOAP no dice nada sobre mapeo de lenguajes porque solamente es un protocolo de transmisión. Así, cualquier vendedor que soporte HTML puede soportar SOAP de forma sencilla (SOAP no exige ni prohíbe comunicaciones basadas en objetos).
- Puesto que utiliza un puerto bien conocido (80), y casi todas las máquinas de Internet entienden las comunicaciones en este Puerto, los firewalls no causarán los típicos problemas de bloqueo y las arquitecturas no necesitarán ser cambiadas para permitir que SOAP se ejecute.
- Puesto que el mecanismo se basa en un intercambio de texto (XML) sobre el protocolo HTTP, es absolutamente independiente de la plataforma de ejecución.
- SOAP puede aprovecharse de las ventajas de utilizar comunicaciones SSL (HTTPS) para asegurar los mensajes que se transmiten.

3.9.5.2 Desventajas

- En su declarada simplicidad, no cubre aspectos como recolección de basura (*garbage collection*), objetos por referencia, activación, comunicaciones HTTP bidireccionales, etc. (en cualquier caso no es una tecnología que se autoproclame como sustitución de CORBA o DCOM, puesto que solamente es un protocolo de envío de mensajes y no constituye un modelo completo de distribución de objetos).

- La generación, serialización y mapeado de SOAP hacen que sea un protocolo con una utilización de recursos más intensa y por lo tanto es más lento que otros protocolos (como IIOP o DCOM).
- SOAP requiere típicamente de 10-15 veces el ancho de banda que otros protocolos (IIOP), porque está orientado a texto y hace que los paquetes sean mucho mayores al ser enviados.
- El mecanismo de tipado de objetos que utiliza es muy débil y no permite una distinción precisa entre algunos de los tipos definidos en los lenguajes de programación.
- Su escalabilidad es incierta, sobretodo por el hecho de no utilizar un puerto independiente. El puerto 80, en ocasiones se convierte en un importante cuello de botella en muchos servidores.
- El hecho de encapsular un protocolo encima de HTTP puede suponer un riesgo de seguridad, puesto que es una puerta abierta (el puerto 80 suele estar accesible) en la mayoría de los ordenadores.
- Es una tecnología realmente nueva (aún se sigue en estado de *draft*) y todavía no ha sido suficientemente probada y validada. No tiene de momento tanto soporte como otras tecnologías.

3.9.6 Resumen

La siguiente tabla resume las principales características de las diferentes tecnologías que se han presentado en este apartado:

	PRINCIPALES VENTAJAS	PRINCIPALES DESVENTAJAS
CORBA	Estándar industrial muy estable. Servicios de objeto. Soporte multilenguaje	Complejidad, eficiencia, transmisión de objetos, sin recolección de basura
JAVA IDL	Parte del núcleo JDK (gratis), abre el mundo CORBA a la plataforma Java	Igual que antes, no implementa muchos servicios CORBA
RMI	Parte del núcleo JDK (gratis), simple para aplicaciones Java-Java, recolección de basura distribuida, paso de objetos	Protocolo no interoperable, sin servicios adicionales, escalabilidad, soporte de los navegadores
RMI-IIOP	Gratis, interoperabilidad con CORBA sin IDL	Requiere el plugin de Java, necesita madurar, la interoperabilidad no siempre es posible
DCE	Implementación de referencia de libre distribución, servicios adicionales	No se adapta bien a OO, necesita puente para Java, sin invocación dinámica
DCOM	Avalado por Microsoft, soporte de herramientas de desarrollo, interop. MS	Originalmente no OO, código no portable, puente para Java
SOAP	Basado en estándares de Internet, transparente a los firewalls, muy flexible	Utilización de recursos, lento, ancho de banda, escalabilidad, soporte, tecnología emergente

Tabla 9. Comparación de tecnologías de distribución de objetos

3.10 Servidores de aplicaciones

Antes de pasar a la descripción de los servidores de aplicaciones se va a introducir la evolución arquitectural que, al aplicarla al esquema de servicios Web que hoy en día se tiene, motiva la aparición de los servidores de aplicaciones como herramienta de ayuda al desarrollo, instalación y mantenimiento de dichos servicios.

Dicha descripción además, completa también la explicación de los diferentes esquemas de interacción en el Web que ya se comentaron en el apartado 2.4.4.

3.10.1 Arquitectura multicapa

El modelo más sencillo (y más antiguo) de arquitectura de aplicaciones, es el ***modelo monolítico*** o de una capa, en el que todos los componentes (aplicación cliente y datos) están integrados en el mismo entorno (típico esquema del *mainframe* o del mini ordenador). Dicha arquitectura cuenta con ventajas muy significativas, derivadas todas ellas de la orientación centralizada que presenta: las aplicaciones con fáciles de gestionar, controlar y de asegurar y los sistemas en general son muy fiables y son capaces de dar soporte a gran cantidad de usuarios.

Sin embargo se trata de un modelo cuya escalabilidad es más que reducida (se limita a las posibilidades de duplicar completamente el entorno) y que además suele ser tremendamente dependiente del sistema operativo y de la máquina en que se ejecuta (si la máquina empieza a sobrecargarse, la única alternativa es comprar otra más potente). Esto hace que en general este tipo de arquitecturas sea muy poco adaptable a los cambios en la tecnología y por lo tanto es un modelo que hace tiempo que empezó a estar en desuso.

Con la llegada de los ordenadores personales, las redes de área local y las bases de datos relacionales, las arquitecturas monolíticas dejaron paso a esquemas más abiertos, basados generalmente en el paradigma cliente servidor (***arquitecturas de dos capas***). A partir de este modelo empiezan a evolucionar las aplicaciones Web (ver Figura 23)

En estas arquitecturas, prácticamente toda la lógica se lleva a los clientes (contando normalmente con una interfaz gráfica bastante potente), de tal forma que los servidores quedan mucho más descargados y normalmente sólo se responsabilizan de la base de datos (por lo que no hace falta recurrir a los grandes ordenadores, con el consecuente ahorro de dinero).

Pese a las ventajas que presenta, como la sencillez, la fiabilidad, la mayor flexibilidad (sobre todo en cuanto a gestión de base de datos se refiere), la mayor accesibilidad a los datos por parte de diferentes aplicaciones, etc., presenta también importantes inconvenientes:

- ***Escalabilidad***: a medida que el número de usuarios crece y dado que cada uno de ellos exige tener una conexión propia con la base de datos, es inevitable que las prestaciones de ésta se degraden considerablemente.
- ***Reutilización***: en las arquitecturas tradicionales de dos capas, la lógica de negocio (o *lógica de aplicación*) se encuentra ubicada normalmente en el cliente, por lo que resulta más complicado reutilizarla.
- ***Fiabilidad***: desde el momento en que la fiabilidad de la aplicación pasa a depender en primer lugar de los ordenadores personales y en segundo lugar de la red y en segundo lugar de los ordenadores personales (considerado como uno de los entornos más difíciles de gestionar), todavía hay muchos casos en los las empresas se resisten a migrar sus aplicaciones críticas a este tipo de arquitecturas.
- ***Mantenimiento de aplicaciones***: como hay que ir distribuyendo los actualizaciones cliente a cliente el mantenimiento se hace considerablemente complicado.
- ***Estructura de la base de datos***: la base de datos, evidentemente se gestiona de una forma más sencilla y está más abierta a la utilización por parte de otras aplicaciones que en el caso monolítico. Sin embargo, hay que tener en cuenta que todas las aplicaciones son absolutamente dependientes de los cambios que haya que hacer en esta base de datos.
- ***Usuarios remotos***: pese a que el modelo puede ser relativamente adecuado para llevarlo a la práctica entre empresas, de cara al usuario final, no suele ser bien recibido el hecho de tener que instalar una aplicación en su ordenador.

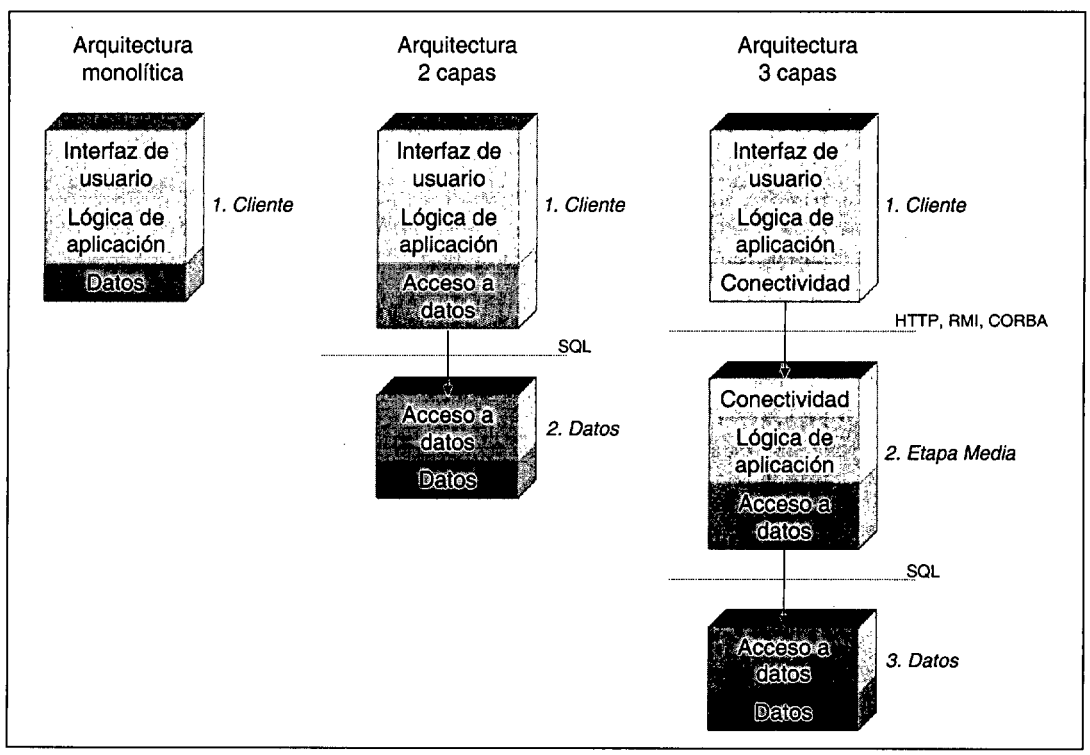


Figura 35. Arquitecturas multicapa

Con el objetivo de tratar de evolucionar el esquema Web hacia algo más potente que el modelo de cliente/servidor basado en HTML y HTTP tan sencillo que se tenía entonces y en vista de que como se apuntaba antes, los clientes son muy reticentes a la instalación de software en sus máquinas (este modelo ya les obliga a instalar al menos el navegador),

la arquitectura evolucionó hacia un esquema modificado de las arquitecturas de dos capas.

El incremento de la potencia de los clientes, primero con Javascript y posteriormente culminado con los applets de Java, aportan verdaderamente todas las ventajas del modelo arquitectural de dos capas al Web (hasta ahora se tenían dos capas, pero a excepción de las limitadas funcionalidades que ofrecían los CGI, ambas capas carecían prácticamente de lógica de ningún tipo y la interfaz gráfica dejaba mucho que desear).

Además vienen también a solucionar tres de las grandes desventajas de estos modelos. Por un lado no se obliga al cliente a instalar ningún programa adicional, porque los navegadores tienen soporte para Java. Además, gracias a las ventajas de portabilidad que ofrece Java, los clientes son totalmente reutilizables. Por último, con respecto al mantenimiento de aplicaciones, desde el momento en que dicha aplicación reside realmente al servidor (es transferida al cliente para ser ejecutada, pero el *código* reside en el servidor), los clientes siempre contarán con la última versión de la misma sin tener que hacer absolutamente nada.

El tercer y último paso en la evolución de este tipo de arquitecturas se alcanza con las ***arquitecturas de tres capas***.

A diferencia de los casos anteriores, aquí la denominación de 'tres capas' hace referencia a los tres principales componentes lógicos de la arquitectura (capa de presentación (cliente), encargada, capa media (servidor de aplicaciones) y capa de datos) y no al número de máquinas del modelo (también recibe la denominación de ***multicapa***, puesto que el servidor de aplicaciones suele implementarse en varias capas que suelen además estar distribuidas en varias máquinas).

Esta arquitectura divide la lógica de la aplicación entre las tres capas (lógica de presentación, lógica de negocio y lógica de acceso a datos). Dichas capas además, se comunican normalmente entre ellas por medio de una interfaz abstracta, por lo que la funcionalidad interna de cada una (y la complejidad asociada) queda perfectamente oculta.

Las principales ventajas que pueden obtenerse de este tipo de diseño son:

- ***Escalabilidad***: este tipo de arquitecturas son muy adecuadas en entornos con un gran número de clientes, puesto que los servidores de aplicaciones pueden distribuirse en diferentes máquinas (distribuyendo así la carga asociada a diferentes tareas) y además las bases de datos no tienen que comprometer una conexión con cada cliente, sino que lo hacen con un número reducido de servidores de aplicaciones.
- ***Reutilización***: además del hecho de que la misma lógica puede ser accedida por gran cantidad de clientes y aplicaciones diferentes, lo cierto es que las ventajas desde la perspectiva de la reutilización llegan más allá del ciclo de vida de una aplicación. De hecho, en este tipo de arquitecturas, más que implementar una aplicación, lo que se hace es crear una serie de componentes capaces de comunicarse entre ellos utilizando una interfaz estándar y abstracta (típica encapsulación que ofrece el modelo de orientación a objetos). Y cada uno de esos

componentes constituye en sí mismo un módulo que puede reutilizarse por completo en otra aplicación totalmente distinta.

- ***Gestión de la lógica de negocio***: las actualizaciones en la lógica de negocio afectan exclusivamente a la capa intermedia de la arquitectura y no hay que ir distribuyendo los cambios entre cada uno de los clientes ni ir adaptando las bases de datos a esos cambios. Además esa capa intermedia ya se ha dicho que suele estar construida a partir de módulo diferentes e independientes, con lo que cambiar una determinada funcionalidad, normalmente implica cambiar solamente uno de esos módulos.
- ***Ocultación de la base de datos***: desde el momento en que el cliente no accede directamente a la base de datos, se pueden hacer en la misma gran cantidad de cambios sin que el cliente perciba absolutamente nada (cambios de estructura, de ubicación, etc.).
- ***Seguridad***: los mecanismos de seguridad pueden ahora implementarse en diferentes niveles (y no solamente en la base de datos). Se puede definir el acceso de los clientes incluso servicio a servicio y además como los clientes en ningún caso acceden a la base de datos, se evita la posibilidad de que accedan a datos no autorizados. La seguridad de la lógica de negocio también se ve beneficiada, pues la capa intermedia suele estar ubicada en servidores centrales más seguros.
- ***Integridad de datos***: todos los accesos a la base de datos se hacen a través de una capa intermedia, lo cual favorece el hecho de que se controle que sólo los datos que se estime que son válidos lleguen a la base de datos.
- ***Mejora de la disponibilidad***: las aplicaciones críticas se ven favorecidas con la posibilidad de proporcionar redundancia de forma sencilla tanto al nivel de los servidores de aplicaciones como al nivel de los servidores de datos.
- ***Especialización de los desarrolladores***: este tipo de arquitectura altamente modular facilita el hecho de que los programadores se especialicen en un determinado tipo de tarea (desarrollo de interfaces gráficas, diseñadores de bases de datos, etc.) e incrementen así su eficiencia.
- ***Infraestructura de computación distribuida***: como parte integral de todo tipo de arquitectura multicapa, se encuentra una infraestructura completa de que facilita la computación distribuida y garantiza un elevado nivel de transparencia en la distribución de los componentes así como diferentes niveles de seguridad, escalabilidad o fiabilidad.

Todas las desventajas de este tipo de arquitectura, pueden concentrarse básicamente en los dos puntos siguientes:

- ***Alta complejidad***: en general es mucho más complicado construir aplicaciones de tres capas que de dos, puesto que a pesar de que las diferentes tecnologías que se han visto en este capítulo se encargan de proporcionar un elevado grado de transparencia a la distribución, la cantidad de detalles y tecnologías diferentes que hay que tener en cuenta y que hay que integrar en una arquitectura de este estilo, hace que esa transparencia no sea sino una ayuda más, en vez de constituirse en la solución definitiva.
- ***Nivel de desarrollo***: los servidores de aplicaciones más importantes del panorama de servicios Web, J2EE y .NET, llevan dos años existiendo, y lo cierto es que todavía no hay demasiadas herramientas que den soporte a este tipo de tecnologías (en el caso de J2EE, por ejemplo no hay más de doce implementaciones y una en el caso de .NET, [J2EEc]). Además, puesto que las tecnologías están todavía en

fase de expansión las versiones aún se suceden a una velocidad considerablemente alta, lo cual complica más aún el tema.

3.10.2 Introducción a los servidores de aplicaciones

Los servidores de aplicaciones como ya se ha dicho son la pieza fundamental de las arquitecturas multicapa puesto que en ellos está ubicada toda la lógica de negocio de la aplicación.

Pese a que en este apartado se dará una descripción de los dos servidores de aplicaciones más importantes (J2EE y .NET) y se proporcionará una comparativa de ambas, se entrará mucho más en el detalle de la arquitectura J2EE, en primer lugar porque es la más difundida y en segundo lugar porque su especificación es más abierta y es la única que se ha definido orientada a constituirse en un estándar.

La especificación de J2EE viene evidentemente definida por Sun, pero a diferencia de .NET, que no es sino un producto de Microsoft, la especificación de Sun puede ser implementada por cualquier empresa que licencie la tecnología (hay más de treinta empresas que ya han obtenido una licencia de J2EE, entre las que se encuentran Borland, Compaq, Fujitsu, Hitachi, HP, IBM, Nec o Nokia).

3.10.3 Java 2 Enterprise Edition (J2EE)

3.10.3.1 Introducción

Una de las razones principales por las que Java se convirtió rápidamente en una herramienta utilizada por gran cantidad de programadores de aplicaciones es su arquitectura portable basada en la máquina virtual Java. Así se contribuyó definitivamente al desarrollo de la tecnología de los clientes Web, que empezaban a estancarse en el HTML y el Javascript como principales lenguajes de desarrollo. Los programadores pueden construir potentes interfaces gráficas para las aplicaciones o los applets clientes que se ejecutan en navegadores Web, independientemente del sistema operativo o la máquina en la que estén ejecutándose.

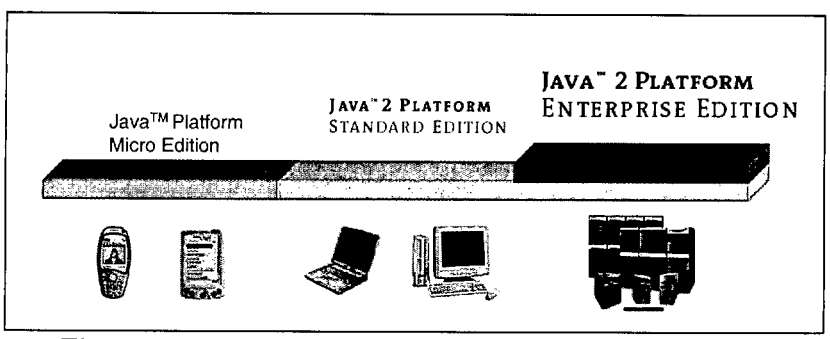


Figura 36. Plataformas de Java para diferentes entornos

Lo que se persigue ahora es aprovechar las facilidades que ofrece este diseño en el lado del servidor, de forma que aplicaciones que sigan el estándar J2EE puedan ser

ejecutadas bajo cualquier servidor de aplicaciones J2EE en cualquier plataforma que soporte el estándar.

Las Java™ 2 Enterprise Edition son un conjunto de APIs estándar de Java definidos por Sun, que han sido fruto de la colaboración de diferentes fabricantes líderes en el desarrollo de software. Sin embargo, es importante destacar que J2EE no es un servidor de aplicaciones en sí mismo. Al igual que ocurre con CORBA, es simplemente una especificación estándar que cualquier servidor de aplicaciones puede implementar (total o parcialmente). Muchos fabricantes de servidores de aplicaciones ofrecen diferentes niveles de soporte J2EE dentro de su línea de productos. Sun proporciona una implementación de referencia (J2EE SDK) que se puede utilizar de forma gratuita para demostraciones, prototipos y con fines educativos. El estándar incluye también una especificación completa de tests para poder asegurar la portabilidad de aplicaciones a través de distintos sistemas capaces de soportar J2EE.

Las APIs que proporciona la última versión de J2EE (1.3), incluyendo también las que se derivan de la compatibilidad con la plataforma J2SE (ver Figura 36) y que permiten desarrollar, configurar y ejecutar aplicaciones en un entorno distribuido son las siguientes (ver [J2EE] y [Kas00]):

- *HTTP / HTTPS*
- *Java™ Transaction API v1.0 (JTA)*
- *RMI-IIOP*
- *JDBC™ API v2.0*
- *Java™ Messaging Service v1.0.2 (JMS)*
- *Java Naming and Directory Interface™ v1.2 (JNDI)*
- *JavaMail™ v1.2*
- *Enterprise JavaBeans™ v2.0 (EJB)*
- *Java™ Servlets v2.3*
- *JavaServer Pages™ v1.2 (JSP)*
- *Java™ API for XML Parsing v1.1 (JAXP)*
- *J2EE™ Connector Architecture 1.0*

Además de estas APIs, la plataforma J2EE proporciona diferentes servicios de nivel de sistema (gestión de transacciones, seguridad, conectividad de clientes, acceso a bases de datos, etc) posibilitando que los desarrolladores puedan centrarse en la lógica de negocio de las aplicaciones y no en los detalles de bajo nivel.

Por último, con respecto al entorno J2EE SDK de Sun (implementación de referencia gratuita) hay que destacar que también incluye software adicional de soporte como el servidor HTTP y HTTPs (el Apache Tomcat) o una base de datos de Cloudscape.

3.10.3.2 Arquitectura J2EE

La Figura 37, muestra la arquitectura en tres capa de J2EE (sin bien la capa media está claramente divida en dos, la capa Web y la capa EJB).

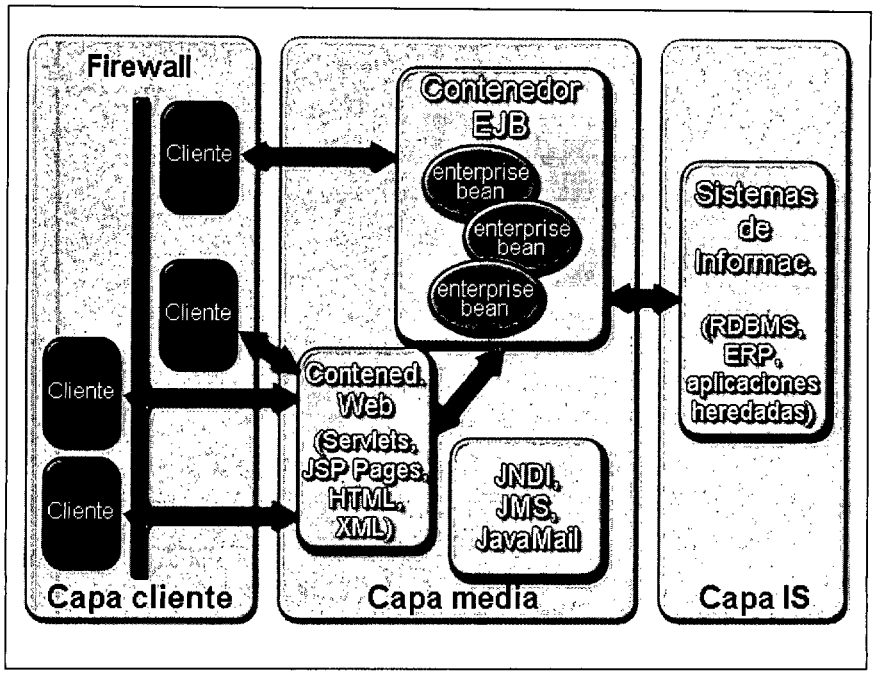


Figura 37. Arquitectura multicapa genérica J2EE [J2EEb]

- ***Capa cliente***: permite que el usuario se comunique con la aplicación J2EE. En esta capa tiene lugar toda la interacción con el cliente y la captura de datos.
- ***Capa Web***: permite que la funcionalidad de la aplicación esté disponible en el Web. Esta capa normalmente se comunica con la capa del cliente utilizando HTTP y su acceso a las otras capas puede hacerse utilizando diferentes protocolos.
- ***Capa EJB (Enterprise JavaBeans™)***: en mitad de la arquitectura J2EE están las Enterprise JavaBeans™ (EJB™) que son responsable de resolver la lógica de negocio y el acceso a los datos de la aplicación. Son componentes reutilizables con los que los desarrolladores pueden codificar prácticamente cualquier funcionalidad de objeto necesaria (mediante beans de sesión o de entidad) y pueden ser accedidos por programas clientes (utilizando RMI) y ser ejecutados en servidores J2EE. Dichos servidores son capaces de gestionar muchos de los aspectos necesarios para poder desarrollar programas de una forma escalable: invocación remota de métodos, balanceo de carga, recuperación transparente de fallos, integración con sistemas de bases de datos heterogéneos, transacciones distribuidas, asociación de recursos, redespliegue dinámico, cierre del sistema transparente, bitácora de eventos, gestión del sistema, multihebras, gestión de comunicaciones asíncrona u orientada a mensaje, ciclo de vida de objetos, gestión de caché, memoria compartida, etc.
- ***Capa de datos (EIS, Enterprise Information System)***: esta capa permite integrar las aplicaciones J2EE con otros sistemas de información de la empresa, como pueden ser las bases de datos, sistemas de gestión de recursos, procesadores de transacciones, etc. El resto de las capas pueden acceder a través de esta capa a las bases de datos utilizando el API JDBC o bien a otros recursos utilizando la tecnología que incorpora J2EE para estos casos *Java Connector Extensión.*

Toda esta arquitectura en capas de J2EE que se acaba de comentar, es muy flexible. Las aplicaciones no tienen por qué utilizar todas las capas y es posible por ejemplo construir arquitecturas de dos capas, con la típica aplicación cliente de Java accediendo

directamente a la base de datos con JDBC e incluso construir aplicaciones de una sola capa (ver Figura 38).

Además la división en estas cuatro capas es exclusivamente funcional, es decir que pueden estar ubicadas todas en la misma red, en el mismo ordenador e incluso en la misma máquina virtual. De hecho, aunque hay diferentes tecnologías como los servlets o las EJBs, que sí pertenecen claramente a una de las capas, otras tecnologías como JMS o la familia de APIs de tratamiento XML (2.2.3.2.5), pueden ser utilizadas en diferentes capas.

3.10.3.3 Detalles de la arquitectura J2EE

La Figura 38 muestra los detalles de la arquitectura J2EE, incluyendo las diferentes posibilidades de conexión entre las diferentes capas, las APIs de la plataforma J2SE que dan soporte a cada capa y los componentes que se ejecutan en cada uno de los denominados 'contenedores' de J2EE (*containers*).

Dichos contendores son precisamente las entidades de nivel de sistema que proporcionan el soporte de ejecución a los componentes, incluyendo todos los servicios añadidos que se han estado comentando, unos provenientes de la plataforma J2SE y otros del propio contenedor (gestión del ciclo de vida, seguridad, gestión de transacciones, capacidades multihebra o gestión de conexiones, ver 3.10.3.3.4).

Precisamente por ser los contenedores los responsables de los servicios de los componentes durante la ejecución de los mismos, para añadir mayor flexibilidad a la plataforma, se permite que los componentes puedan ser adaptados de forma declarativa (es decir sin necesidad de recompilar nada) en el momento en el que se transfieren a los correspondientes contenedores. Dicha adaptación (denominada despliegue, *deployment*, de aplicaciones) es una novedad muy interesante en esquema de desarrollo de software pues permite que los ajustes finales de instalación del servidor de aplicaciones en una determinada plataforma, no exijan la modificación del código y se puedan hacer utilizando una interfaz gráfica de configuración (el resultado es lo que llama *descriptor*, que es un fichero XML con el contenido del mencionado despliegue).

Para finalizar con la descripción de la arquitectura de J2EE, se recorrerán los diferentes contenedores que proporciona la plataforma, comentando los detalles que cada uno de ellos lleva asociados.

3.10.3.3.1 Contenedores del cliente

Los clientes de la plataforma J2EE pueden ser bien clientes Web, o bien aplicaciones independientes.

Los primeros estarían compuestos a su vez de dos partes, las páginas Web dinámicas generadas por los componentes Web que se ejecutan en su correspondiente contenedor Web dentro del servidor de aplicaciones (contienen HTML, XML, etc.) y el navegador Web, que recibe dichas páginas y las interpreta (ver 2.4.2).

Además, las páginas Web que recibe el navegador, pueden también incluir applets, que ejecutándose en la máquina virtual del navegador Web, pueden acceder a la plataforma J2EE e incluso asumir parte de la lógica de negocio de la misma.

Sin embargo, el modelo de cliente recomendado por la especificación es el que se basa en los clientes generados por los componentes de la capa Web, es decir clientes ligeros (*thin clientes*) basados en lenguajes de marcado (HTML, WML, etc.) que no exigen la instalación de ningún plugin en el cliente y que además permiten separar perfectamente las funciones del desarrollo del diseño de páginas Web y de la lógica provista por la aplicación que se mantiene en el servidor.

Las aplicaciones Java se ejecutan también en la máquina del cliente (mediante el *Java Runtime Environment*) y proporcionan a los clientes la posibilidad de gestionar tareas que requieran una interfaz más potente que la que puede obtenerse con un determinado lenguaje de marcado. Estas aplicaciones por tanto no requieren la utilización de la capa Web para acceder a las EJBs de la capa intermedia, e incluso son capaces de acceder a las bases de datos por ellas mismas. Sin embargo, en el caso de que la aplicación así lo requiera, también son capaces de abrir conexiones HTTP y acceder a los componentes de la capa Web.

Por último, hay que comentar que tanto en los clientes como en los servidores se pueden también incluir componentes basados en la tecnología de componentes JavaBeans (ver [Sun02f]), que no están considerados sin embargo componentes J2EE y que se utilizarían para gestionar el flujo de datos entre la aplicación cliente o el applet correspondiente y los componentes que se ejecutan en el servidor (o entre los componentes del servidor y la base de datos).

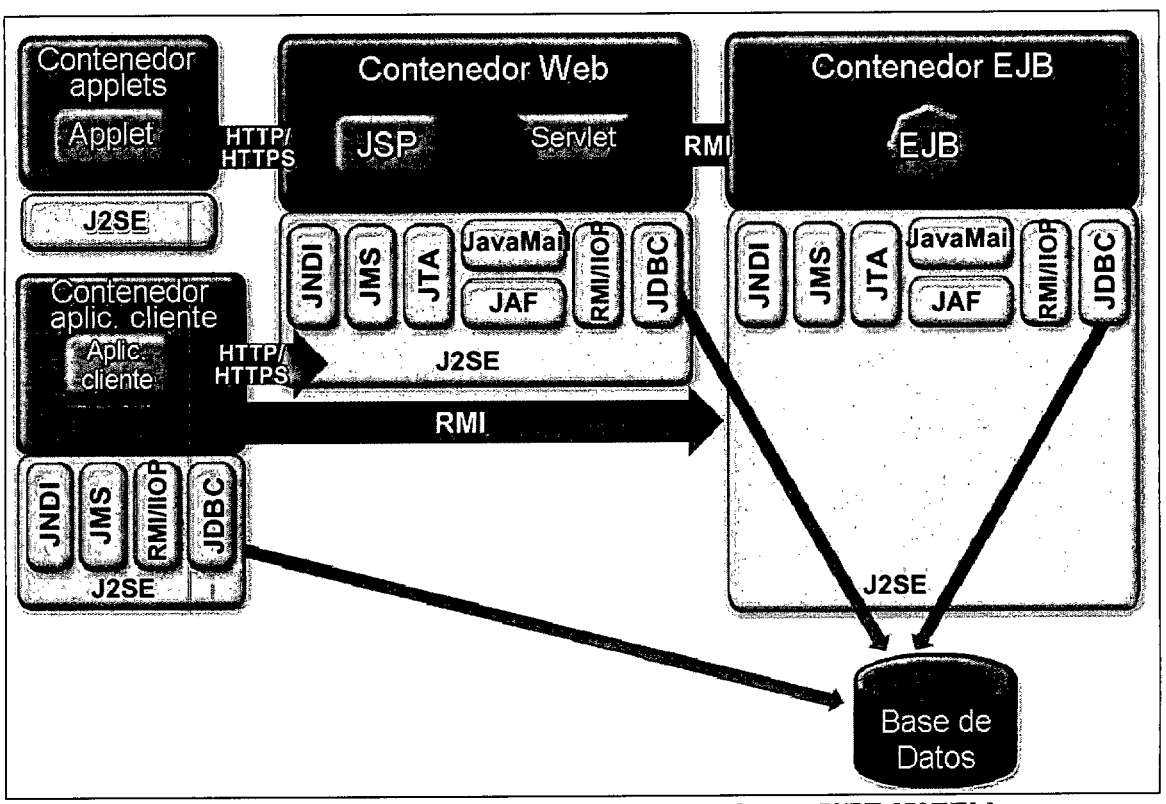


Figura 38. Arquitectura detallada de la plataforma J2EE [J2EEb]

3.10.3.3.2 Contenedor Web

El contenedor Web constituye el entorno de ejecución para los ficheros JSP y los servlets (ver [JSP] y [Servlet]), dotándolos de las APIs y servicios que se han comentado anteriormente (ver también la Figura 38).

JavaServer Pages

Una JSP es un documento de texto que describe cómo procesar una petición para crear una respuesta. Un documento JSP puede contener plantillas para formatear la salida con el aspecto de un documento Web, lo cual no afecta a las componentes activas de la página, y permite una separación de los elementos dinámicos de la interfaz gráfica. Una JSP contendrá también elementos y *scriptlets* (pequeños fragmentos de código Java que directamente se codifican en la página JSP) para generar contenido dinámico. La mayoría de las JSP debería utilizar tecnología de JavaBeans o enterprise beans para realizar el procesado complejo que exija la aplicación. JSP pueden ampliarse mediante el desarrollo de acciones a medida encapsuladas en librerías.

Servlets

Los servlets son un mecanismo portable e independiente de plataforma o servidor, que permite generación de contenido dinámico y por tanto amplían la funcionalidad del servidor Web. Recibida la petición de un cliente, generan automáticamente contenido HTML o XML como respuesta y lo envían de vuelta. Dicho contenido generado, puede provenir bien de alguna decisión tomada por el propio servlet o bien del resultado del procesamiento ejecutado en la capa de las EJBs como consecuencia de que el servlet les haya transferido los datos de la petición del cliente. De la misma forma que en las JSPs, para mantener la filosofía de separación funcional en toda la arquitectura es recomendable que dejen la labor de ejecución de la lógica de negocio a la capa de las EJBs.

Servlets vs. JSP

Las páginas JSP están basadas en texto y orientadas a presentación y permiten desarrollar servlets de forma sencilla. Las JSP ofrecen todas las ventajas de los servlets y cuando se combinan con las JavaBeans o las EJBs, permiten también separar la lógica de contenido y de presentación.

La ventaja de separar el contenido de la lógica de presentación es que la persona que se encarga de actualizar el aspecto de las páginas no tiene porqué entender el código Java y de la misma forma, la persona que actualiza las JavaBeans o las EJBs no tiene porqué ser un experto en el diseño de páginas Web. Tal y como se comentó en el capítulo anterior, las JSP como los servlets son actualmente preferidos a los CGIs, porque son independientes de la plataforma de ejecución, introduce mayor sobrecarga y permite acceder de forma sencilla a los parámetros que se pasan del cliente al servidor.

Se pueden utilizar JSP junto con las clases JavaBeans para definir plantillas y construir un sitio Web a partir de páginas con el mismo formato gráfico. Las clases JavaBeans hacen la transformación de datos de forma que las plantillas no necesitan tener código Java alguno y pueden ser mantenidos por un editor de HTML.

A la hora de elegir entre la utilización de servlets o JSPs, hay que considerar que los servlets son herramientas de programación muy apropiadas para funcionalidades de bajo nivel que no requieran actualizaciones frecuentes mientras que las JSP están más orientadas a presentación y es un mecanismo declarativo para asociar el contenido dinámico y la lógica de la aplicación.

Teniendo una aplicación Web simple utilizando una página JSP, se puede asociar el contenido con la lógica de la aplicación usando etiquetas a medida o *scriptlets* en vez de JavaBeans. Las etiquetas definidas se empaquetan en librerías y se importan en la página JSP.

3.10.3.3.3 Contenedor EJB

Las instancias de las *Enterprise Java Beans* (EJB) se ejecutan dentro de contenedores EJB, que son entornos de ejecución que controlan a las beans y les ofrecen importantes servicios de nivel de sistema. Al no tener que desarrollar estos servicios, es posible centrarse mejor en el desarrollo de la lógica de las beans. Los servicios proporcionados por dicho contenedor son:

- Gestión de transacciones.
- Seguridad.
- Conectividad remota de clientes.
- Gestión del ciclo de vida.
- Gestión de conexiones a bases de datos.

EJB's

Las *enterprise beans* son componentes de servidor escritas en Java que contienen la lógica de negocio de la aplicación. Por ejemplo, el cliente de un banco invocaría los métodos `ingresar_dinero` y `extraer_dinero` sobre la enterprise bean que represente a su cuenta.

Hay tres tipos de enterprise beans: entity beans (de entidad), session beans (de sesión) y *message driven beans* (basadas en mensajes).

Entity beans

Una *entity bean* representa un objeto que lleva asociado algún mecanismo de persistencia, como por ejemplo una base de datos. Una entity bean podría representar a un cliente, que podría almacenarse como una fila en la tabla de usuarios de una base de datos relacional (o en una base de datos de objetos, un fichero o cualquier otro tipo de almacenamiento). El mecanismo de almacenamiento depende de la tecnología particular de implementación de EJB aunque la implementación de referencia (J2EE SDK) utiliza una base de datos relacional (Cloudscape).

La persistencia de una bean puede ser gestionada bien por la entity bean por sí misma o bien por el contenedor EJB. En el primer caso (*bean-managed persistence*), es necesario escribir código adicional para gestionar su acceso a las bases de datos (podría incluir en la bean comandos SQL para acceder a una base de datos relacional usando JDBC o separar dichos comandos en una clase auxiliar de acceso a datos de objeto, *Data Access Object* o DAO). En el caso de que la persistencia sea gestiona por el contenedor

(*container-managed persistence*) el contenedor EJB se encarga de gestionar automáticamente las llamadas de acceso a las base de datos.

Session bean

Una *bean de sesión* representa a un cliente en el servidor J2EE. Dicho cliente se comunica con el servidor invocando los métodos pertenecientes en una session bean (`ingresar_dinero`, `sacar_dinero`, etc.). Cuando el cliente termina, su correspondiente *bean de sesión* termina también, es decir, las beans de sesión no son persistentes.

Comparación entre beans de sesión y entidad

Aunque ambos tipos de beans se ejecutan en el mismo tipo de contenedor EJB, son muy diferentes. La siguiente tabla resume las principales diferencias entre las beans de sesión y de entidad.

	Objetivo	Acceso compartido	Persistencia
Session Bean	Ejecuta alguna tarea por el cliente	Puede tener un cliente	No persistente. Cuando termina el cliente, su session bean también.
Entity Bean	Representa un objeto de negocio que existe de forma persistente.	Puede ser compartida por muchos clientes.	Persistente. Aunque el contenedor EJB termine, la bean permanece en la base de datos

Tabla 10. Comparación de beans de sesión y de entidad

Message driven bean

Este tipo de beans (MDB) son componentes que se encargan de procesar mensajes asíncronos generados mediante JMS (*Java Message Service*), API que forma parte de las J2EE. Este tipo de beans aparece solamente a partir de la última versión de la plataforma (J2EE 1.3) y aunque son responsables de hacer todo el procesado que se ha comentado, es el contenedor el que se encarga de gestionar su entorno (propiedades transaccionales, asentimiento de mensajes, concurrencia, etc.). Este último detalle de la concurrencia supone una ventaja muy importante con respecto a los clientes tradicionales de JMS, de tal forma que una MDB es capaz de recibir cientos de mensajes y procesarlos todos de forma simultánea.

Pese a que es una bean completa y recibe el mismo tratamiento por parte del contenedor que las otras dos, este tipo de bean no tiene interfaz de componente. El motivo es que estas beans no son accesibles mediante RMI y solamente se puede interaccionar con ellas utilizando JMS.

Componentes JavaBeans y Enterprise Beans

Aunque el nombre es muy similar y ambos componentes están escritos en Java, los componentes JavaBeans y las enterprise Java beans no son, ni mucho menos, lo mismo. Los componentes JavaBeans definen una convención para hacer clases Java configurables por herramientas de diseño, permitiendo que esas herramientas enlacen los

objetos mediante eventos. Las *enterprise beans* se utilizan para implementar servicios transaccionales multiusuario.

3.10.3.3.4 Servicios

Conectividad

Conjunto de conexiones a la base de datos

La obtención de una conexión con la base de datos es un recurso muy costoso, pues además de exigir bastante tiempo (puede llegar a ser del orden de varios segundos), es posible que el número máximo de conexiones esté limitado (éste de hecho era uno de los inconvenientes que antes se planteaba para las arquitecturas de dos capas). Para resolver estos problemas el contenedor gestiona un conjunto de conexiones a las bases de datos (*database connection pool*), de forma que pueda obtenerse rápidamente una conexión de dicho conjunto cuando sea necesario. Cuando la bean libera la conexión se retorna al conjunto y puede ser reutilizada por otra bean.

Conectividad remota de clientes

El contenedor gestiona de forma transparente las comunicaciones de bajo nivel entre clientes y EJBs. Después de que una EJB se haya creado, un cliente invoca métodos sobre ella como si estuviesen en la misma máquina virtual.

Transacciones

Cuando un cliente invoca un método en una EJB, el contenedor interviene automáticamente para gestionar la transacción, por lo que no hay que incluir código especial al efecto en la EJB (el código requerido para controlar transacciones distribuidas puede ser muy complejo). En vez de escribir y depurar código complejo, simplemente hay que declarar las propiedades transaccionales de la bean un su correspondiente fichero descriptor (etapa declarativa del *deployment* que se comentaba antes). El contenedor leerá el fichero y gestionará las transacciones de las EJBs de forma automática.

Seguridad

El contenedor sólo permite la invocación de métodos en las EJBs a los clientes autorizados. Cada cliente puede pertenecer a un grupo particular y cada grupo puede tener también determinados permisos de ejecución.

Los grupos y los métodos que puede invocar, se declaran en el descriptor de la EJB, por lo que no es necesario codificar rutinas de seguridad concretas en las beans para reforzar la seguridad.

Gestión del ciclo de vida

Durante su tiempo de vida, una *enterprise bean* pasa por diferentes etapas. El contenedor crea la *enterprise bean*, la mueve del espacio de recursos disponibles a estado activo y finalmente la elimina. Aunque el cliente dispone de métodos para realizar las mismas acciones por su cuenta, el contenedor realiza estas funciones de forma transparente.

3.10.3.4 Ciclo de implementación de aplicaciones J2EE

Los desarrolladores de J2EE lo que en definitiva escriben son componentes de aplicación, que son módulos auto-contenidos que pueden incluir applets, aplicaciones cliente ligeras, servlets, JSPs o EJBs.

Todos estos componentes se empaquetan en forma de archivos Java (JAR) o archivos Web (WAR) y posteriormente, tras proceder a desplegarlos en el servidor de aplicaciones, se obtienen archivos de aplicaciones de empresa (EAR).

Un ejemplo de aplicación J2EE podría tener un formulario HTML para obtener los datos de usuario, un servlet para recibir los datos y procesarlos y EJBs para almacenar la información en la base de datos. El fichero WAR estaría formado por el formulario HTML y el servlet, el JAR por la EJB y ambos se encapsulan en un fichero EAR que se despliega al final para poner la aplicación en funcionamiento en el servidor.

El modelo de programación de aplicaciones es muy flexible, permitiendo por ejemplo que un fichero WAR pueda incluir páginas JSP auto-contenidas que implementen una aplicación por sí solas, o una aplicación compuesta por páginas JSP combinadas en un fichero EAR.

Para escribir componentes de aplicación J2EE, un desarrollador necesita obtener un producto J2EE de algún proveedor (ver [J2EEc]). Los proveedores son típicamente vendedores de sistemas operativos, sistemas de bases de datos, servidores de aplicaciones o servidores Web que implementan la plataforma J2EE de acuerdo a la especificación. Sun por su parte, ofrece de forma gratuita su implementación de referencia.

La idea de la portabilidad de J2EE, pretende conseguir que si todos los componentes desarrollados son conformes al estándar J2EE, se podrán ejecutan en cualquier servidor de aplicaciones.

3.10.4 .NET

3.10.4.1 Introducción

.NET [.NET], es un servidor de aplicaciones desarrollado por Microsoft para crear servicios Web basados en XML. Dichos servicios permiten que las aplicaciones se comuniquen y compartan datos a través de Internet independientemente del sistema operativo o del lenguaje de programación (.NET es un servidor de aplicaciones para el entorno Windows, pero los servicios Web que se pueden crear con él, son independientes).

La plataforma incorpora gran cantidad de tecnologías basadas en XML y en estándares de Internet, que proporcionan soporte para el desarrollo, gestión y operación de servicios Web. Más específicamente, de entre las áreas en las que Microsoft espera contribuir activamente con su plataforma se podrían destacar:

- ***Clientes***: los clientes podrán ser absolutamente heterogéneos: PCs, portátiles, estaciones de trabajo, teléfonos, consolas de juegos, etc. La idea es dotar a estos

dispositivos de compatibilidad con .NET, proporcionándoles el software necesario como para que puedan utilizar los servicios Web. De esta forma el usuario puede acceder a sus datos independientemente de su localización o el tipo de cliente que utilice.

- ***Servicios***: los servicios Web XML permiten que las aplicaciones se comuniquen independientemente del sistema operativo sobre el que se ejecuten y el lenguaje de programación que utilicen puesto que las tecnologías de soporte que utilizan, vienen especificadas por organizaciones de estandarización públicas como el W3C. Además, con los servicios Web XML, las aplicaciones no sólo pueden compartir datos sino que tienen también acceso a las posibilidades que ofrecen otras aplicaciones independientemente de cómo hayan sido construidas unas y otras. Compartir datos mediante XML permite que puedan ser independientes unas de otras y a la vez puedan referenciarse unas a otras colaborando para llevar a cabo diferentes tareas.
- ***Servidores***: los servidores de aplicación .NET, incluyendo la familia de servidores Windows 2000 constituyen una amplia infraestructura para desarrollar y gestionar servicios Web XML (ver [.NETa])
- ***Herramientas***: Visual Studio.NET y el entorno .NET de Microsoft pretenden proporcionar una solución completa para que los desarrolladores implementen, desplieguen y ejecuten servicios Web XML y ASP.NET. Visual Studio.NET es la herramienta de desarrollo multi-lenguaje de Microsoft, construida específicamente para .NET. Aprovechando la potencia de lenguajes como Visual Basic o Visual C++, permite desarrollar de forma sencilla aplicaciones .NET.

3.10.4.2 Arquitectura

La Figura 39 muestra los diferentes estratos en los que puede dividirse la arquitectura de la plataforma .NET y que son los que se describen a continuación.

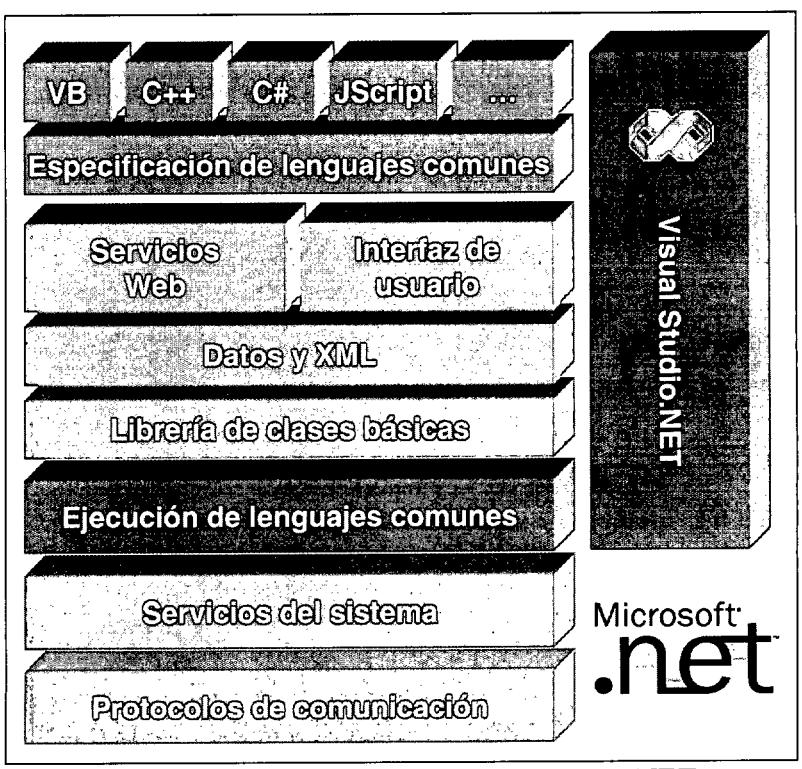


Figura 39. Arquitectura de la plataforma .NET

- ***Protocolos de comunicación***: en lugar de definir nuevos protocolos que cubran las necesidades de COM/DCOM o de CORBA, lo que se hace es recurrir a la utilización de protocolos estándar basados en XML sobre HTTP, como SOAP (ver 3.9.5), UDDI [UDDI] o WSDL [WSDL]. De esta forma se garantiza la integración de las aplicaciones generadas por esta plataforma con el resto de aplicaciones Web que utilicen estos mismos estándares.
- ***Servicios del sistema***: pese a no formar parte de la plataforma .NET, sí forman parte de Windows, el sistema operativo sobre el que se ejecuta .NET. Por lo tanto es posible proporcionar diferentes servicios como transacciones (COM+), mensajería (MSMQ), acceso a datos (MSDB), servicio de directorios (Active Directory), etc.
- ***Infraestructura de lenguajes comunes***: la implementación que se hace del estándar ECMA-335 (*Common Language Infrastructure*, [CLI]), por medio de los diferentes componentes de la capa de ejecución de lenguajes comunes (CLR, *Common Language Runtime*) y la de especificación de lenguajes comunes (CLS, *Common Language Specificación*), es uno de los aspectos más innovadores y potentes de la arquitectura NET. La implementación permite dar soporte a la interoperatividad entre diferentes lenguajes de programación de alto nivel, posibilitando que cualquiera de ellos se ejecute en la plataforma y acceda a las librerías que esta proporciona. Para ello se implementan diferentes aspectos definidos en el estándar, como un sistema de tipos comunes, un sistema de metadatos y un entorno de ejecución completo que se inspira en muchas de las características de la máquina virtual Java (gestión de hebras, gestión de memoria, recolección de basura, seguridad, compiladores JIT, un cargador de clases, etc.). Siguiendo con las similitudes que se observan con respecto a Java, en el entorno CLR lo que realmente se ejecuta es un lenguaje intermedio de *bytecodes* IL (*Intermediate Language*) que hace posible la interoperabilidad de la que se ha hablado, en la medida en que existan compiladores (cuya construcción se ve favorecida por los diferentes componentes de la CLI) de lenguajes que cumplan el estándar de tipos (CTS, *Common Type System*) a ese leguaje intermedio (en general, para cualquier lenguaje de programación externo a la plataforma, habrá que definir un subconjunto o superconjunto de funcionalidades si se quiere utilizar).
- ***Librerías básicas (.NET Base Framework)***: es una colección de librerías a la que tienen acceso todos los lenguajes .NET. Esto por un lado garantiza que todos los lenguajes de la plataforma tienen una implementación común pero por otro lado hace que las funciones de muchos lenguajes sean ahora incompatibles (algunas de las funcionalidades asociadas a determinados lenguajes de programación se han incorporado en estas librerías por lo que, por ejemplo, funciones que existen en Visual Basic, no existen en VisualBasic.NET y hay que acceder a ellas usando las librerías básicas).
- ***Servicios de datos***: la nueva versión de los componentes ADO (*Active Data Object*) de acceso a datos (denominada ADO+ o ADO.NET), presenta ya por separado el modelo de datos y los datos en sí mismos, con un esquema basado en XML y en SOAP para el mantenimiento y el intercambio de datos.
- ***Servicios Web e interfaz de usuario***: los servicios Web y los formulario de Windows, proporcionan la interacción de la plataforma con el usuario (sea el usuario *real* o bien un componente, otro servicio Web, etc.). Se posibilita por un lado el rápido desarrollo de interfaces y por otro la versatilidad que ofrecen todas las tecnologías asociadas a los servicios Web (ver 2.5).

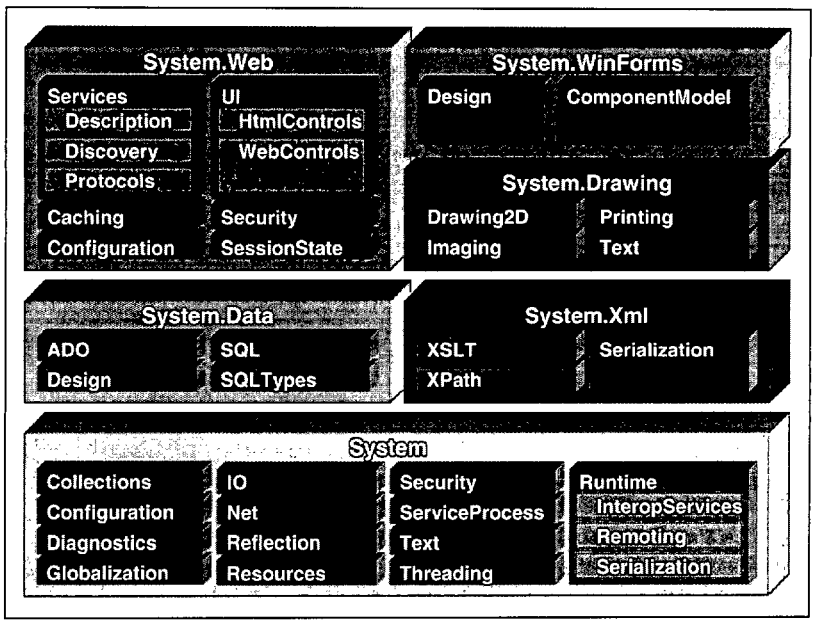


Figura 40. Algunas librerías básicas de soporte a los lenguajes .NET

- ***Lenguajes***: además de las características que desde la perspectiva de interoperabilidad, librerías de lenguajes, etc., se proporciona también una implementación del estándar ECMA-334 (*C# Language Specification*, [C#]), que incorpora las principales ventajas de Java y C++.
- ***Visual Studio .NET***: es el entorno de desarrollo que proporciona Microsoft para manejar todos los aspectos de la arquitectura de forma gráfica e intuitiva (proporciona transparencia desde el punto de vista del desarrollo a SOAP, a WSDL, etc.).

3.10.5 Comparativa

Hoy en día las comparativas entre estos dos tipos de tecnologías son muy abundantes, puesto que son sin duda las dos plataformas del momento, en cuanto a desarrollo de aplicaciones orientadas a comercio electrónico basado en Web.

La mayoría de las comparativas no obstante, suelen mostrar una inclinación muy marcada por alguna de las tecnologías, por lo que en muchos casos (sobre todo cuando se hace referencia a determinadas valoraciones cuantitativas de las plataformas) es recomendable aceptarlas con cierto grado escepticismo.

Por otro lado, las dos tecnologías son aún demasiado nuevas como para que existan criterios de peso que eliminen a una de las dos por algún motivo y además evolucionan tan rápidamente y se incluyen mejoras y correcciones con tanta velocidad que a veces incluso lo que se consideraba una desventaja competitiva, con la siguiente versión se consigue una ventaja. Esto ocasiona también que determinadas comparaciones queden obsoletas en poco tiempo, por lo que a la hora de analizarlas, siempre hay que considerar con respecto a qué versión se hacen las comparaciones (en el caso de J2EE, habrá también que considerar con respecto a qué implementación, puesto que en función ésta, las prestaciones pueden variar de manera muy importante).

La siguiente lista muestra una serie de características comparadas de ambas tecnologías que provienen de agregar diferentes puntos de vista de diferentes artículos y estudios recientes (ver por ejemplo [VR01] o [Ses01]):

- ***Portabilidad***: por definición en el diseño de ambas tecnologías, mientras que .NET es una plataforma totalmente dependiente de Microsoft y su sistema operativo Windows (es un producto más de Microsoft), J2EE está especificado como un estándar, que cualquier empresa puede implementar y además que garantiza la interoperabilidad entre las diferentes implementaciones, independientemente del sistema operativo en el que se ejecute cada una de ellas, etc. Sin embargo, lo cierto es que pese a que efectivamente hay gran cantidad de implementaciones de J2EE (y sólo una de .NET), la interoperabilidad entre ellas no termina de estar conseguida del todo.
- ***Lenguaje de programación***: la plataforma J2EE sólo soporta como lenguaje de programación Java (aunque es posible la interoperabilidad con componentes escritos en otro lenguaje es mediante CORBA, JCA o JNI). .NET sin embargo está específicamente diseñado (CLR) para dar soporte a cualquier lenguaje de programación (en principio se espera dar soporte a prácticamente todos los grandes lenguajes, excepto Java). Sin embargo, para que un lenguaje sea compatible con .NET es necesario definir un nuevo super/sub lenguaje basado en el anterior que sea compatible con el CLS de .NET y además debe existir el correspondiente compilador que pase de este lenguaje a IL.
- ***Soporte para servicios Web***: ambas tecnologías son en principio muy adecuadas para el desarrollo de servicios Web. Pese a que el soporte para el desarrollo de contenido XML ha tardado en llegar por parte de Sun, actualmente se ya proporciona una completa serie de APIs para el tratamiento de XML (en desarrollo sigue todavía, por ejemplo, un API específico para el tratamiento de WSDL o para UDDI, pero en cualquier caso pueden utilizar sin mayor problema las APIs genéricas de tratamiento XML). .NET soporta perfectamente el tratamiento de XML, SOAP, WSDL o UDDI, pero sin embargo no da soporte a ebXML, que parece erigirse en la alternativa más importante para ampliar los servicios Web hacia la integración comercial global (Microsoft da soporte a su propia plataforma Biztalk, que utiliza un versión propietaria de SOAP).

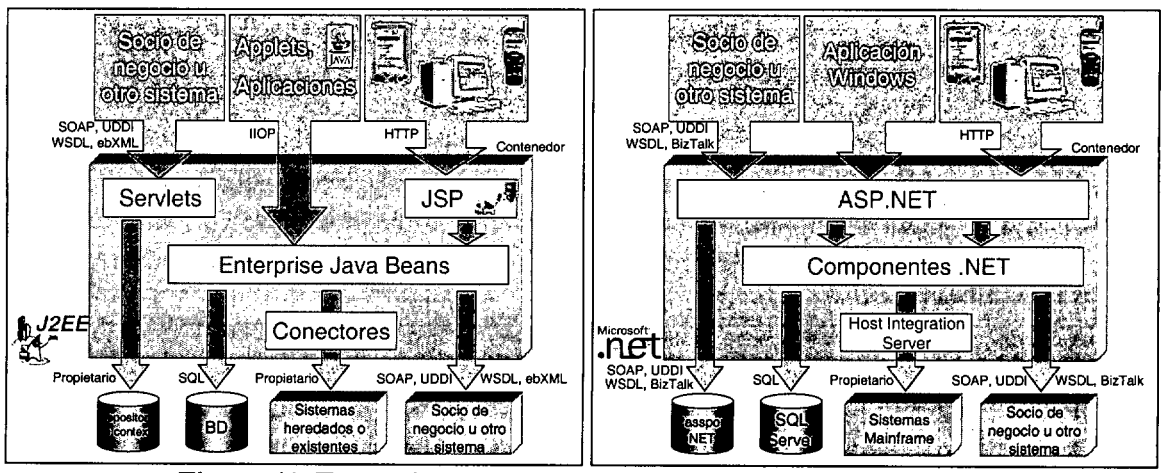


Figura 41. Tratamiento de servicios Web mediante J2EE y .NET

- ***Interfaz de usuarios***: la tecnologías ASP.NET de Microsoft, es absolutamente independiente del dispositivo de acceso del cliente por lo que una determinada interfaz de usuario puede convertirse a otro tipo de interfaz alternativa sin necesidad de rescribir código alguno (en el caso de J2EE esto no es así y es necesario escribir código específico en función del dispositivo de acceso final).
- ***Características comerciales***: desde el punto de vista de marketing y de mercado y pese a que J2EE cuenta con el apoyo de cerca de 50 empresas, lo cierto es que la capacidad de Microsoft para publicitar su producto es mucho mayor. Por otro lado el anuncio del soporte a servicios Web llegó bastante antes por parte de la plataforma de Microsoft que de la de Sun.
- ***Madurez de la tecnología***: pese a que las tecnologías en las que se basa J2EE presentan una madurez y estabilidad mayor que las de .NET, no es menos cierto que las primeras implementaciones de J2EE tardaron en aparecer más tiempo que las de .NET. En cualquier caso, ambas tecnologías pueden considerarse todavía muy nuevas y todavía tienen mucho que demostrar en gran cantidad de aspectos. En J2EE, es relativamente novedoso el soporte a los servicios Web, la arquitectura JCA o la persistencia automática de las EJB. En el caso de .NET, el nuevo CLR ha obligado a rescribir parte del núcleo de la plataforma y tanto el lenguaje C# como el soporte para servicios Web son nuevos.
- ***Reutilización de aplicaciones existentes***: la plataforma J2EE permite la reutilización de sistemas heredados mediante tecnologías como JMS (para integrar sistemas basados en mensajería), servicios Web (para cualquier tipo de sistema Web existente), CORBA (para interaccionar con componentes CORBA escritos en otros lenguajes), JNI (para cargar librerías de código nativo y poder ejecutarlas localmente) o JCA (para escribir adaptadores que entiendan las comunicaciones con sistemas existentes). .NET soporta prácticamente lo mismo con sus correspondientes tecnologías asociadas (Host Integration Server 2000, COM Transaction Integrator, Microsoft Message Queue o Biztalk Server 2000). La única diferencia podría estar en la versatilidad que propiciará JCA (aunque es una tecnología que todavía está adquiriendo madurez) que no parece tener réplica entre las herramientas de Microsoft.
- ***Precio***: en este sentido, la flexibilidad que ofrece J2EE presenta considerables ventajas. El precio de la plataforma variará en función de la implementación que se elija (hay gran cantidad de ellas disponibles) y el hardware y sistema operativo de soporte (J2EE también está abierto en este sentido). Las opciones podrían ir desde una implementación de iPlanet ejecutándose sobre Solaris a la implementación de jBoss sobre Linux. En el caso de Microsoft, no hay elección posible.
- ***Prestaciones***: desde el punto de vista arquitectural, ambas tecnologías ofrecen las mismas perspectivas en cuanto a prestaciones (y ambas exigen muchos recursos para ser ejecutadas). En este sentido lo cierto es que las diferentes estadísticas y comparativas que pueden verse por Internet, parecen mostrar mejores prestaciones para .NET. Sin embargo hay que tener siempre en cuenta en este tipo de comparativas la implementación de J2EE que se utiliza en el análisis (mientras que .NET sólo está implementado por Microsoft, J2EE viene implementado por multitud de fabricantes y cada implementación evidentemente tiene prestaciones muy diferentes).
- ***Herramientas de desarrollo***: en este sentido el entorno de desarrollo que ofrece Microsoft con su Visual Studio.NET integrando absolutamente todas las tecnologías necesarias para la implementación de aplicaciones, no tienen réplica

en el mundo de J2EE (aunque también tiene a su disposición gran cantidad de herramientas desarrolladas por importantes fabricantes como Borland, IBM o Visual Café).

- ***Escalabilidad***: ambas tecnologías ofrecen características de escalabilidad similares (incluyendo balanceo de carga, por ejemplo).

3.11 Conclusión

En este capítulo se ha empezado por describir numerosas tecnologías que si bien arquitecturalmente eran de lo más diverso y procedían de diferentes iniciativas de todo género, todas ellas presentaban un objetivo común: proporcionar un marco de integración para aplicaciones en entornos distribuidos.

Desde principios de los años noventa, la tecnología que lideró el sector del middleware ha sido sin duda alguna CORBA.

La cantidad de implementaciones que empezaron a aparecer a partir de la especificación tanto de los protocolos como de los servicios que constituyen la arquitectura CORBA, hicieron que pronto se afianzase en el sector de la distribución de objetos.

De la misma forma que ahora abundan comparaciones sobre J2EE y .NET, hace unos años lo que se planteaba era la clásica disyuntiva de elegir entre CORBA o DCOM. En ambos casos una de las tecnologías venía auspiciada por Microsoft y en ambos casos la otra definía una especificación abierta a la que había que proporcionar implementaciones (en el caso de CORBA mucho más marcado, evidentemente, pues lo que había detrás no era una empresa sino una organización no lucrativa).

Sin embargo, el hecho de que CORBA fuese una tecnología (probablemente la única) absolutamente independiente de la plataforma de ejecución (soporta prácticamente cualquier sistema operativo y todos los principales lenguajes de alto nivel) y del vendedor que implemente la especificación (si la implementación se hace conforme al estándar, queda garantizada la compatibilidad entre diferentes ORBs), hizo que la mayoría de las empresas adoptasen CORBA cuando el objetivo era lograr interoperabilidad entre determinados componentes de la infraestructura de la organización (evidentemente cuando la integración se restringía a entornos Windows, la opción era DCOM).

A partir de mediados, finales de los noventa, el comercio electrónico y las tecnologías Web empiezan a difundirse. Es una época en la que Java ya se ha implantado como lenguaje de programación de las aplicaciones del momento, pero en la que todavía no existe ninguna alternativa seria al tipo de esquema middleware de la potencia que plantea CORBA.

Empieza entonces un proceso lento y progresivo de implementación de plataformas CORBA que pretendían dar soporte al lenguaje Java y lograr así la unión dos tecnologías marcadas por su flexibilidad y por la portabilidad que ofrecían. Se trata también de acercar la tecnología CORBA al mundo de los servicios que se empezaban a dar por

Internet y que de alguna manera se habían ido separando de los esquemas un tanto pesados que podían ofrecer tanto CORBA como DCOM.

Al mismo tiempo que se van desarrollando plataformas CORBA con soporte para Java, la tecnología de distribución de objetos de Java (RMI), continúa evolucionando y a finales de los noventa, dado que no termina de lograrse una integración entre CORBA y Java plenamente satisfactoria y que sea completamente adaptable a los modelos de negocio que empiezan a aparecer, RMI empieza a constituirse en una seria alternativa a CORBA.

Sin embargo RMI no cuenta con el soporte de los servicios que ofrece CORBA y utiliza un protocolo propietario para proporcionar el transporte de los datos. Hay además, gran cantidad de aplicaciones basadas en CORBA todavía, que no resultaría sencillo migrar a otros entornos.

En 1999, un proyecto conjunto entre Sun e IBM termina con la aparición de RMI sobre IIOP, lo cual por un lado le da a RMI el empuje de estandarización que necesitaba y por otro aporta el único detalle que faltaba para unir definitivamente las tecnologías CORBA y Java.

El auge de las tecnologías XML y la redefinición de los servicios Web como fruto de la interrelación entre esas tecnologías es la oportunidad que aprovecha Microsoft para empezar a apostar por la apertura que tanto éxito le ha dado a tecnologías como CORBA y Java. En su plataforma .NET y pese a que internamente sí se recurre a COM+ como mecanismo de comunicación entre componentes, el tipo de servicios que se pueden generar, incluyen aquellos en los que la definición se realiza en WSDL, la localización se realiza mediante UDDI y el intercambio de datos mediante SOAP.

De la misma forma Sun en su plataforma J2EE, incorpora la posibilidad de utilizar unas APIs específicas que facilitan el tratamiento de documentos XML y por lo tanto la creación de servicios Web también se ve facilitada (internamente recurre a RMI sobre IIOP para la comunicación entre los componentes del servidor).

Actualmente ambas tecnologías .NET y J2EE están en plena fase de expansión y pese a las numerosas comparaciones que se hacen entre ellas, aún es pronto para saber si habrá una ganadora, o por el contrario ambas convivirán, tal y como ocurrió en el pasado con otras tecnologías rivales.

Desde el punto de vista de CORBA, parece que últimamente ha empezado a caer en desuso en el marco de los servicios Web y del comercio electrónico, puesto que estas nuevas tecnologías se han diseñado específicamente para simplificar el desarrollo de aplicaciones dentro de ese contexto.

Visto que la heterogeneidad que se tenía antes como proveniente de los diferentes lenguajes y plataformas hardware, parece haberse superado con las plataformas middleware, ahora es posible que el problema de la interoperabilidad se plantee entre aplicaciones que hayan sido diseñadas con un middleware determinado y hayan por tanto ligado a él su posible evolución (en principio no ocurriría en el caso de los servicios Web, pero sí en el resto de aplicaciones que por ejemplo se encarguen de aportar la lógica suficiente como para tratar dichos servicios).

El OMG ha detectado este problema y en línea con el desarrollo que se hizo en el caso de CORBA, hace poco más de un año se está trabajando en la arquitectura y especificación de MDA (*Model Driven Arcuitecture*, [MDA]) cuyo objetivo último es lograr la interoperabilidad del software de diferentes aplicaciones, independizándolo incluso de de la plataforma middleware que se esté encargando de proporcionarle la transparencia de distribución.

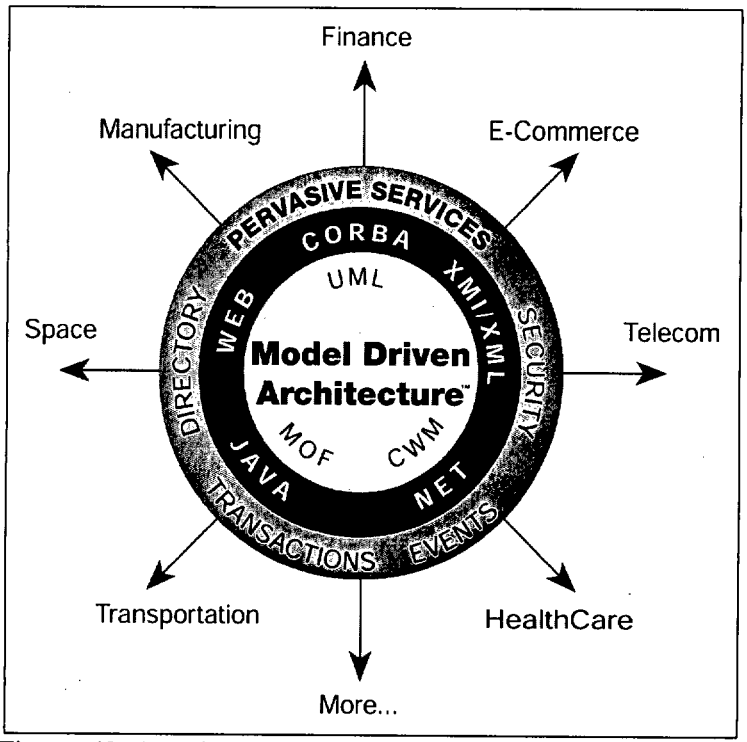


Figura 42. Arquitectura MDA definida por el OMG [MDA]

Cualquiera que sea el resultado de este trabajo de especificación, aún está lejos de poder aplicarse realmente y mucho más aún de que las empresas se planteen la implantación de la tecnología dentro de sus esquemas de desarrollo, en los que de momento la tendencia que se observa es hacia la utilización de los potentes servidores de aplicaciones de los que se ha hablado.

Capítulo 4

Arquitecturas de comercio electrónico

4.1 Introducción

Hace no más de cinco años, cuando se hablaba de arquitecturas, plataformas o herramientas de comercio electrónico, casi siempre se estaba hablando de sencillos esquemas de navegación Web que, en el mejor de los casos, implementaban algún mecanismo de integración con plataformas de pago.

Poco a poco fueron evolucionando las posibilidades tecnológicas, sofisticándose la compra por catálogo, incluyendo la metáfora del carrito de la compra virtual, etc., pero aún así, cualquiera que fuese el esquema utilizado, en general no pasaba del modelo de negocio de una tienda virtual más o menos avanzado.

A medida que el comercio electrónico entra en esa etapa de rápida implantación de la que se ha hablado y las empresas punto-com empiezan a proliferar, también empiezan a aparecer numerosos productos que se venden bajo el nombre de plataformas o servidores de comercio electrónico y que permiten generar esquemas de tiendas virtuales integradas con gran rapidez.

Tras la caída del modelo de negocio de las empresas virtuales, quedan atrás gran cantidad de esquemas de negocio, que probablemente no han tenido éxito como consecuencia de las circunstancias del comercio electrónico en una época determinada,

pero que merece la pena aprovechar de cara a construir las bases de un mercado electrónico global amplio, que dé cabida a la mayor cantidad de opciones posibles.

En esta línea se empezó a trabajar, con el objetivo de desarrollar soluciones de negocio electrónico integral y todo ese esfuerzo está empezando a dar sus frutos.

Actualmente están empezando a aparecer soluciones de integración a nivel de aplicaciones de empresa (*Enterprise Application Integration*) y productos que empiezan a dar soporte al modelo avanzado de plataformas de intermediación, como los servidores de aplicaciones orientados al desarrollo de servicios Web que se vieron en capítulos anteriores. Sin embargo, estos productos son todavía demasiado específicos como para ser aplicables a un entorno de mercado global.

Pese a que se ha hablado de que tecnologías como WSDL, WSFL o XLANG, pretenden generalizar y extender el concepto de servicios Web y aportar ciertos valores semánticos que van más allá de la simple catalogación de servicios, lo cierto es que por un lado estas tecnologías están todavía lejos de su madurez y por otro, la extensión del modelo de servicios Web al ámbito empresarial exigen una definición mucho más precisa de las transacciones de negocio, esquemas de datos y flujos de documentos.

*El **objetivo de este capítulo** es presentar diferentes propuestas que se están planteando, más que para cubrir un determinado modelo de negocio, para abordar esquemas de comercio y negocio electrónico global (incluyendo aproximaciones B2B), mediante la definición de mecanismos de interoperabilidad y colaboración genéricos de alto nivel. En primer lugar se repasarán las iniciativas más importantes del sector y posteriormente se irán detallando algunas de las propuestas más relevantes.*

4.2 Iniciativas

Antes de describir algunos de los principales marcos arquitecturales de comercio electrónico, este apartado hace un recorrido por las iniciativas más importantes del sector, pasando por organizaciones de estandarización, consorcios empresariales, iniciativas de ámbito privado, proyectos de investigación financiados por administraciones públicas, etc.

Es importante destacar que la selección de iniciativas se ha hecho entre aquellas organizaciones que han realizado aportaciones de interés en el ámbito de las arquitecturas de comercio electrónico, más que entre organizaciones que aunque quizá tengan también gran relevancia en el sector, no son sino meros usuarios de las alternativas tecnológicas que otras ofrecen.

Asociada a las descripciones de cada una de esas iniciativas, se hace una breve mención a sus propuestas más reseñables, algunas de las cuales se comentarán más a fondo en apartados sucesivos.

4.2.1 Estándares europeos

La Unión Europea ha reconocido mediante una directiva a tres organizaciones como competentes en el área de la estandarización tecnológica en Europa: el CEN (*Comité Européen de Normalisation*, [CEN]), el CENELEC (*Comité Européen de Normalisation Electroniqué*, [CENELEC]) y el ETSI (*European Telecommunications Standards Institute*, [ETSI]).

Entre las tres organizaciones preparan estándares en diferentes sectores, motivados fundamentalmente tanto por peticiones de la industria, como por peticiones de la propia Comisión Europea de cara a desarrollar su legislación interna.

Figura 43. Principales grupos con iniciativas en arquitecturas de comercio electrónico

Aunque las competencias y ámbitos de actuación de cada organización están perfectamente especificados, es inevitable que en algunas áreas como telecomunicaciones, tecnologías de la información, etc. se solapen. Por ello, se define un comité especial para coordinar los intereses y actuaciones de las tres instituciones.

En materia de comercio electrónico, los estándares europeos se articulan a través de una rama del CEN creada en 1997, para centrarse en actividades relacionadas con la sociedad de la información: el ISSS (*Information Society Standarization System*, [ISSS]). Dicho comité se creó con la filosofía de desarrollar normas en un entorno sujeto a cambios rápidos y constantes, por lo que se encuentra a medio camino entre los procesos abiertos basados en mecanismos formales de estandarización y los procesos de normalización rápida que exige el mercado.

Los dos grupos que lideran las iniciativas en comercio electrónico en el ISSS son el WS/eBES (*e-business Board for European Standarization*, [eBES]) y sobre todo el

WS/EC (*Electronic Commerce*, [EC]). El primero está centrado actualmente en coordinar los esfuerzos europeos por seguir las tendencias marcadas por la utilización de XML como base de interoperabilidad de las arquitecturas de comercio electrónico (fundamentalmente *ebXML*, ver 4.4). El segundo está encargado del seguimiento de los diferentes proyectos y propuestas de estandarización en curso. En relación con arquitecturas de comercio electrónico (propuestas por el WS/EC) destacan:

- *Building Blocks* [BBlocks]: es un proyecto ya finalizado que se centraba en el análisis de los diferentes bloques que hacen falta para construir una arquitectura de comercio electrónico. También aborda temas como la interoperabilidad entre plataformas ofreciendo un análisis sistemático de las relaciones entre los aspectos técnicos, legales y de estandarización en el comercio electrónico.
- *Architectures and Models for Electronic Commerce* [ECarch]: proyecto en curso cuyo objetivo es la identificación, seguimiento y análisis de los diferentes modelos y arquitecturas sobre comercio electrónico propuestas por diversas organizaciones.
- *E-Commerce Integration Meta-Framework* [ECIMF]: proyecto en curso que busca la definición de un estándar de modelado de entornos de interoperabilidad que permita definir fácilmente y de manera homogénea cuáles son los requisitos de interoperabilidad de un determinado marco de comercio electrónico.

4.2.2 ISOC/IETF

La Sociedad de Internet (*Internet Society*, [ISOC]) es una institución sin ánimo de lucro fundada en 1992, que cuenta (mediados de 2001) con más de 150 miembros de diferentes organizaciones y 6000 miembros individuales de más de 100 países. Su principal objetivo es fomentar la expansión de Internet por lo que, entre otras cosas, financia y proporciona soporte legal a los diferentes grupos tecnológicos: IETF, IESG, IRST, IAB e IANA.

De ellos, es el IETF (*Internet Engineering Task Force*, [IETF]) el que tiene por misión el desarrollo (identificación de problemas técnicos, propuesta de soluciones, etc.) de nuevas especificaciones de estándares para Internet, que se publican en forma de RFC (*Request For Comments*). El IETF es una gran comunidad internacional de diseñadores, operadores, fabricantes e investigadores de tecnologías de red involucrados en la evolución de la arquitectura de Internet. Aunque los primeros RFCs datan de 1969, el IETF como tal fue oficialmente establecido en 1986 por el IAB (*Internet Architecture Board* [IAB], encargado de ir fijando los intereses generales de Internet) y cuando en 1992 se creó la ISOC y el IAB pasó a depender de ella, el IETF por extensión, también pasó a depender de la ISOC.

El trabajo técnico en el IETF está gestionado por el IESG (*Internet Engineering Steering Group*) y dividido en áreas de especialización (aplicaciones, Internet, gestión, encaminamiento, sub-ip, seguridad, transporte y servicios de usuario) cada una con diferentes grupos de trabajo organizados (en total más de 120 grupos). En lo que respecta al comercio electrónico, el área más involucrada es la de 'aplicaciones' y dentro de ella destacan dos grupos:

- *Internet Open Trading Protocol* [IETFa]: es el grupo del IETF más directamente vinculado con el comercio electrónico y se encarga de desarrollar una especificación de un entorno de interoperabilidad para el comercio electrónico en Internet.
- *Electronic Data Interchange-Internet Integration* [IETFb]: centrado en aspectos relacionados con el intercambio electrónico de datos en Internet (EDI), evalúa las diferentes propuestas tradicionales existentes y estudia los requisitos desde el punto de vista actual de Internet y las posibles sinergias con las tecnologías emergentes (sobre todo con XML).

4.2.3 Unión Europea

La Unión Europea [UE], a través de la Comisión Europea financia una serie de programas destinados principalmente a fomentar la investigación y el desarrollo en unas determinadas áreas.

En el Cuarto Programa Marco (1994-1998) dentro del área de "*Tecnologías de la información y las comunicaciones*" se crearon tres diferentes programas de apoyo a la investigación: el ACTS [ACTS], el ESPRIT [ESPRIT] y el TELEMATICS [TELEMATICS].

De estos tres programas, el que más proyectos presenta desde el punto de vista de arquitecturas de comercio electrónico y sistemas de intermediación es el ACTS. Dicho programa se estableció con el objetivo genérico de apoyar la investigación en tecnologías y servicios de comunicación avanzados para facilitar el desarrollo económico y la cohesión social en Europa.

Asociados al programa ACTS ha habido más de 200 proyectos de investigación, estructurados en seis dominios y perteneciendo casi todos los proyectos de comercio electrónico al quinto de ellos (*Service Engineering, Security and Communications Management*). A dicho dominio pertenecen proyectos como ABS, ABROSE, COBRA, OSM o SEMPER.

Actualmente está finalizando el Quinto Programa Marco, que se encarga de cubrir las exigencias de desarrollo tecnológico en la Unión Europea en el período que comprende los años 1998 a 2002. En dicho programa se establecen una serie de áreas temáticas quedando el sector de las tecnologías de la información encuadrado en el área 2: *user-friendly information society* (IST).

Dicho área fija cuatro acciones tecnológicas (*key actions*), quedando la segunda de ellas explícitamente vinculada al comercio electrónico: *New Methods of Working and Electronic Commerce* [ISTKAII]. Esta acción tecnológica cuenta ya con más de 300 proyectos asociados.

4.2.4 CommerceNet

CommerceNet [Comm], es un consorcio fundado en Sillicon Valley en 1994 con el objetivo de utilizar Internet como entorno global para la implantación y el desarrollo del comercio electrónico.

Hoy en día, es uno de los consorcios más importantes y cuenta con más de 500 miembros en todo el mundo. Además cuenta con miembros independientes, tanto organizativa como económicamente, a nivel nacional en gran cantidad de países (incluido España [CNE]).

Los proyectos más destacables de CommerceNet son:

- *RossetaNet* [Rosetta]: este proyecto de CommerceNet ha dado pie a una organización sin ánimo de lucro con el mismo nombre que el proyecto, formada por importantes empresas de la industria de los ordenadores, fabricantes de semiconductores y cadenas de suministros. Su propósito es el de promocionar y desarrollar un conjunto de procesos de negocio estándares (principalmente basados en XML) para potenciar el funcionamiento electrónico de estas cadenas de suministros.
- *Open Buying in the Internet* [OBI]: su objetivo es proporcionar un marco estándar para un comercio B2B seguro e interoperable y con un interés especial en la automatización de procesos caracterizados por un elevado número de interacciones con bajos intercambios económicos.
- *E-Commerce architecture* [eCo]: esta especificación tiene por misión el descubrimiento de negocios que proporcionan unos determinados productos y servicios y una vez localizados, determinar cómo puede interactuarse con sus sistemas de comercio electrónico.
- *Universal Marketplaces* [UM]: este proyecto, todavía en una fase muy preliminar, pretende proponer una arquitectura de comercio electrónico basándose en la que se presentaba en eCo. Dicha arquitectura esta orientada a soluciones B2B con el objetivo de crear un mercado interoperable en el que se puedan conectar de forma transparente vendedores, clientes, suministradores y socios de negocio.

4.2.5 OMG

El OMG (*Object Management Group* [OMG]), es una organización no lucrativa creada en 1989 por un consorcio inicial de 11 empresas (entre ellas 3Com Corporation, American Airlines, Canon, Hewlett-Packard, Philips Telecommunications, Sun Microsystems y Unisys Corporation).

Actualmente el consorcio incluye más de 800 empresas con el compromiso de crear especificaciones para el desarrollo de software industrial y bajo unos criterios de excelencia técnica, viabilidad comercial e independencia (neutralidad).

Las especificaciones del OMG, tienen el propósito común de crear un conjunto de componentes reutilizables mediante la estandarización de objetos software y de interfaces para computación distribuida de objetos. Más concretamente, uno de los principales objetivos de las especificaciones es el de apoyar el establecimiento de CORBA (ver 3.4) como plataforma middleware universal (al OMG pertenecen también conocidas especificaciones como OMG IDL, IIOP, OMA, Domain Facilities, UML o la más reciente MDA).

Desde el punto de vista organizativo, está estructurado en tres comités que engloban todos los grupos de trabajo. El Comité de Plataformas Tecnológicas (*Platform Technology Committee*, PTC) y el Comité de Dominios Tecnológicos (*Domain Technology Committee*, DTC) son los responsables de las labores técnicas, mientras que el Comité de Arquitectura (*Architectural Board*, AB) se encarga de garantizar la consistencia e integridad de los trabajos realizados en el PTC y DTC.

Para liderar las iniciativas y especificaciones relacionadas con el comercio electrónico, se ha definido un grupo de trabajo dentro del DTC: el EC-DTF (*Electronic Commerce Domain Task Force*, [ECDTF]) cuyos resultados más relevantes son:

- *Negotiation Framework* RFP (*Request for Proposals*): sus resultados fueron adoptados a finales de 1999 como ampliación de una especificación genérica sobre objetos de negocio definida por otro grupo de trabajo (el *Common Enterprise Model DTF*). Los tres entornos definidos en la especificación (*Document*, *Community* y *Collaboration*) adaptan al mundo del comercio electrónico los patrones definidos por el CEM-DTF.
- *Public Key Infrastructure* RFP: pendiente de su adopción definitiva, proporciona interfaces estándares para acceder a autoridades de certificación y registro permitiendo la gestión de claves públicas mediante Internet.
- *Registration and Discovery RFP*: en un estado más preliminar que el anterior, pretende proporcionar una extensión a la iniciativa del CEM-DTF previamente citada, para búsqueda y registro.

4.2.6 UN/CEFACT

El UN/CEFACT (*United Nations Centre for Trade Facilitation and Electronic Business*, [UN/CEFACT]) es un comité de las Naciones Unidas establecido en 1996 por la Comisión Económica Europea de las Naciones Unidas ([UNECE]), en respuesta a una necesidad de reconocer las contribuciones tecnológicas del sector y de reutilizar los recursos existentes de forma eficiente. Como parte del UN/ECE, responde de sus acciones ante el ECOSOC (*Economic and Social Council of the United Nations*, [ECOSOC]), el comité de las Naciones Unidas más importante en lo que respecta a economía, comercio y desarrollo.

El objetivo de UN/CEFACT es mejorar las aptitudes de las organizaciones relacionadas con negocio, con comercio o con la administración, pertenecientes a cualquier tipo de economía (desarrollada, en desarrollo o en transición) en cuanto a la realización de intercambios de productos y servicios de una forma efectiva y contribuir en general al crecimiento del comercial global (reducir la burocracia, incrementar la

transparencia y reducir los costes de los procesos comerciales, mejorar el flujo de datos en el comercio electrónico, desarrollar una red de instituciones de soporte).

Desde el punto de vista de organización interna, las diferentes recomendaciones del UN/CEFACT son emitidas por los cuerpos tecnológicos y grupos de trabajo que comanda. Uno de esos grupos está específicamente diseñado para encargarse del comercio electrónico: *Electronic Commerce Ad hoc Working Group* [ECAWG].

Actualmente sin embargo, una de sus propuestas más interesantes desde el punto de vista del comercio electrónico, es la iniciativa ebXML ([ebXML], ver 4.4) que patrocina junto con OASIS (ver 4.2.7) y que está recibiendo muy buena acogida.

4.2.7 OASIS

La organización para el avance de estándares de información estructurados (*Organization for the Advancement of Structured Information Standards*, [OASIS]) es un consorcio internacional sin ánimo de lucro cuyo principal objetivo es la creación y el fomento de estándares de interoperabilidad para la industria basados en estándares públicos de estructuración de información como XML, SGML y CGM.

Es consorcio se creó en 1993 y actualmente cuenta con más de 170 miembros de la industria (incluyendo importantes empresas del sector telemático como IBM, Cisco Systems, Adobe Systems, BEA Systems, Hewlet-Packard, Intel, IONA, Microsoft, Netscape/AOL, Oracle, etc.). A escala mundial es considerada como líder en la aplicación de XML al desarrollo de estándares industriales.

El trabajo técnico de OASIS, coordinado por diferentes Comités Técnicos, está generalmente relacionado con una de las siguientes categorías: aplicaciones industriales verticales (desarrollo de aplicaciones XML o SGML, DTDs, hojas de estilo, etc. que puedan usarse en industrias verticales específicas), entornos de negocio horizontal (desarrollo de especificaciones para el intercambio electrónico de información de negocio), interoperabilidad (desarrollo de especificaciones y estándares para definir cómo otros estándares pueden cooperar), pruebas de conformidad (especificación de pruebas para evaluar la conformidad con determinados estándares).

Actualmente, una de las áreas más activas es la que ha llevado junto con UN/CEFACT (ver 4.2.6) al patrocinio de ebXML.

Dicha iniciativa pretende proporcionar una infraestructura basada en XML que permita la utilización global de información de negocio de una forma interoperable, segura y consistente.

4.2.8 Open Applications Group

El Open Application Group [OAG], es un consorcio no lucrativo formado en 1995 por numerosas empresas líderes en el sector de desarrollo de aplicaciones software. Inicialmente el consorcio empezó con ocho empresas y cuenta con más de treinta en la

actualidad (American Software, Electronic Industry Data Exchange Association, CODA Finaltials, eBiz, Oracle, PeopleSoft, Rational, etc.).

Su principal objetivo es facilitar y promover la integración de componentes de aplicaciones software de negocio para la empresa a bajo coste.

Para ello, en 1995 desarrolló una arquitectura y una serie de especificaciones englobadas bajo el nombre genérico de *Open Application Group Integration Specification* [OAGIS]. La utilidad de dicha arquitectura ha sido convenientemente probada desde entonces y actualmente los objetivos del consorcio están centrados en:

- Continuar enriqueciendo su modelo arquitectural.
- Especificar y publicar contenidos XML en consonancia con su objetivo original: fomentar la adopción de un estándar unificado para el negocio electrónico y la interoperabilidad de aplicaciones software (software B2B y A2A).

Es importante destacar que el consorcio no está involucrado en el desarrollo de software de ningún tipo, sino que se centra exclusivamente en las especificaciones, manteniéndose neutral e independiente con respecto a los mecanismos *middleware* de transporte.

4.2.9 Iniciativas empresariales

Pese a que en este sector hay multitud de iniciativas y muchas empresas están desarrollando sus propias arquitecturas que directamente acaban traduciéndose en productos comerciales, son de especial interés las iniciativas de Sun Microsystems y de Microsoft Corporation, por el impacto que sus propuestas pueden tener en el resto de tecnologías que ambas empresas apoyan.

Sun Microsystems hizo su aportación hace unos años en el entorno de las tecnologías Java (*Java EC Framework*). Su objetivo era el de proporcionar una infraestructura completa para implementaciones de soluciones de comercio electrónico basadas en Internet y en el lenguaje de programación Java. Como resultado se obtuvieron un conjunto de herramientas Java que incluían componentes reutilizables de comercio electrónico. Actualmente sólo se están potenciando ya determinados componentes de esta arquitectura y la tendencia de Sun en el sentido de definición arquitectural se limita a la formación de coaliciones empresariales de apoyo a ciertos productos o a la definición de APIs que den soporte a soluciones que ofrecen otros (todas las APIs que se están definiendo para gestionar documentos XML facilitan mucho la utilización de tecnologías como ebXML, por ejemplo).

Por parte de Microsoft, su aportación más importante a las arquitecturas de comercio electrónico es Biztalk. Esta arquitectura está especialmente pensada para adoptar una tecnología muy específica, XML, como mecanismo de intercambio de mensajes de negocio que sean conformes a un formato de datos común sobre una red de comunicaciones como Internet. No hace más hincapié en el tema sin embargo, que no sea la definición de mensajes SOAP dentro del marco del comercio electrónico. La idea de Microsoft con este esquema es la de proporcionar una vía de expansión hacia el comercio inter-empresarial y de mercado electrónico a su plataforma .NET (ver [.NET]).

4.3 BizTalk Framework

4.3.1 Introducción

En la actualidad los protocolos de transporte seguros empiezan a ser lo suficientemente maduros como para que puedan ser aplicados al comercio electrónico y por tanto posibiliten y relancen la ejecución de transacciones de una forma eficiente, segura y automática.

Habitualmente dicho automatismo en las transacciones llevadas a cabo por una empresa se ha logrado en las comunicaciones entre las diferentes líneas internas del negocio, aplicaciones de productividad y gestión del conocimiento, aplicaciones utilizadas por los clientes o socios, o los servicios proporcionados por sus proveedores corporativos.

El coste de conseguir dicha eficiencia y automatismo entre procesos internos es similar al que se plantea para interacciones entre empresas o negocios diferentes. Sin embargo, en estos casos aparecen una serie de nuevos factores que hay que considerar:

- Hace falta un lenguaje flexible lo suficientemente rico y universal como para especificar, empaquetar, publicar y distribuir ***información estructurada*** y no estructurada entre aplicaciones o negocios diferentes.
- Hace falta un lenguaje para especificar y ejecutar transformaciones y ***reglas de conversión de información*** entre los formatos utilizados por las diferentes aplicaciones de las diferentes empresas.
- Se necesitan ***protocolos neutrales*** independientes de la plataforma de nivel de aplicación para posibilitar las interacciones automáticas entre empresas diferentes.
- Se necesitan mecanismos neutrales para proporcionar ***servicios de seguridad de mensajes*** (integridad, privacidad y no repudiación).

Tanto XML como los lenguajes basados en esquemas XML (*XML Schema*), proporcionan la solución buscada en la forma de un conjunto de tecnologías muy potente y con una barrera de entrada considerablemente baja. Estos lenguajes permiten estructurar e intercambiar información de una forma independiente de plataformas o tecnologías middleware.

El resultado es que gran cantidad de iniciativas lideradas por empresas o cuerpos de estandarización han empezado a implementar y adaptar los esquemas XML al sector del comercio electrónico para facilitar las comunicaciones entre aplicaciones y negocios diferentes.

Tanto el lenguaje XML como los lenguajes basados en esquemas XML, conforman un conjunto de potentes tecnologías con una barrera de entrada relativamente baja, que permiten describir e intercambiar información estructurada entre aplicaciones o negocios colaboradores con un formato independiente de la plataforma. Actualmente, tanto iniciativas industriales como diferentes cuerpos de estandarización están adoptando XML o lenguajes basados en esquemas XML para especificar sus vocabularios o sus modelos de datos.

Además, el hecho de que se haya extendido con tanta rapidez, ha llevado a que se implementen gran cantidad de soluciones propietarias con el objetivo final de lograr interoperabilidad entre terceros. Sin embargo, ninguna de estas soluciones reúne todos los requisitos necesarios como para resolver los problemas de utilización de entornos complejos, lo cual ha hecho que los usuarios al final afronten de forma individual e independiente sus problemas de interoperabilidad.

El entorno Biztalk, liderado por Microsoft, pretende solucionar estos problemas de interoperabilidad con una aproximación neutral tanto desde el punto de vista de la plataforma como tecnológico. La idea es proporcionar una especificación para el diseño y desarrollo de soluciones de mensajería basadas en XML para comunicaciones entre aplicaciones y empresas. A bajo nivel utiliza tecnologías estándares como pueden ser HTTP, XML, MIME o SOAP.

Es importante destacar que Biztalk no pretende abordar todas las facetas del comercio electrónico B2B (como aspectos legales, recuperación frente a errores, especificación de procesos de negocio concretos, etc.). El entorno Biztalk proporciona una serie de mecanismos básicos requeridos en casi todos los intercambios de comercio B2B. Se espera que otras especificaciones y estándares compatibles con la especificación Biztalk se desarrollen para cubrir aspectos específicos de un dominio o aplicación.

4.3.2 Arquitectura

En la Figura 44 se pueden se pueden apreciar prácticamente todos los componentes de la arquitectura Biztalk.

Las entidades finales de la arquitectura Biztalk son las aplicaciones, que son las encargadas de emitir y consumir documentos de negocio con los adaptadores necesarios para ello. Dichas aplicaciones envían sus documentos de negocio a servidores BFC (*Biztalk Framework Compliant servers*) que son los encargados de encapsularlos dentro de mensajes Biztalk.

La estructura de cada mensaje, unidad mínima de intercambio entre servidores BFC, depende del tipo de mecanismo de transporte (HTTP, SMTP, etc.) y en ocasiones incorpora cabeceras específicas sobre dicho mecanismo en el mensaje.

En cualquier caso, todo mensaje, debe contener al menos un documento Biztalk que defina la semántica del mensaje dentro del entorno Biztalk. Además puede contener otros mensajes adjuntos, alguno de los cuales puede ser a su vez un documento Biztalk (aunque serán tratados como cualquier otro documento XML, sin semántica especial) y otros pueden ser imágenes, ficheros comprimidos, etc. (los ficheros adjuntos pueden enviarse también de forma independiente o simplemente ser referenciados).

Los documentos Biztalk son mensajes SOAP 1.1 que contienen en el cuerpo documentos de negocio (uno o más) y en la cabecera etiquetas específicas de Biztalk (Biztags) que dotan de semántica específica a los mensajes.

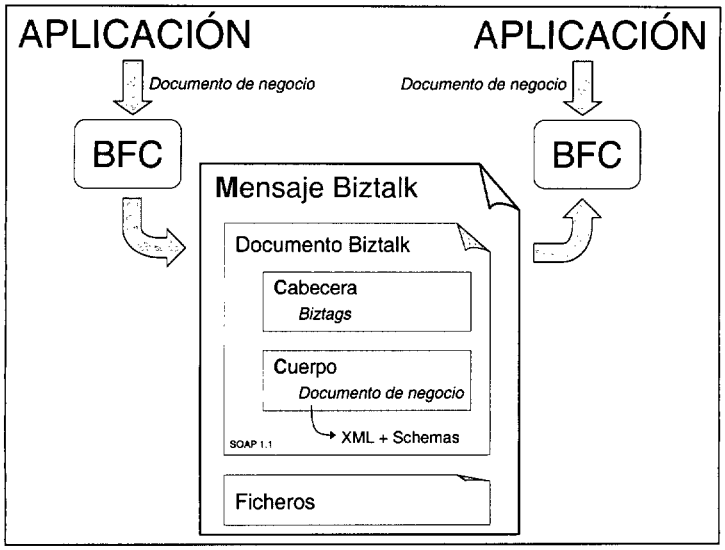


Figura 44. Arquitectura de mensajes Biztalk

Los documentos de negocio, generados por las aplicaciones, son documentos XML bien formados que contienen datos de negocio (orden de compra, factura, previsión de ventas, etc.). Los documentos se describen basándose en metadatos que se engloban en esquemas XML ([XSD1] y [XSD2]). Dichos esquemas contienen las etiquetas necesarias para definir transacciones, tal y como hayan acordado las entidades participantes. Además, permiten crear sistemas que procesen automáticamente documentos de negocio o parseadores que validen automáticamente la compatibilidad con el esquema en tiempo de ejecución. Las organizaciones pueden publicar sus esquemas en librerías Biztalk o mediante otros medios.

La versión 2.0 de Biztalk es una actualización de la versión 1.0 que incluye considerables mejoras en algunos puntos: enlaces de transporte, entrega de mensajes fiable, codificación MIME para ficheros adjuntos, seguridad basada en S/MIME. Además la versión 2 se ha visto influenciada por gran cantidad de tecnologías recientes como la versión 1.1 de SOAP o las estructuras y tipos de datos de XML Schema.

La versión 2.0 incorpora también mejoras en la semántica de las etiquetas de Biztalk (Biztags) de tal forma que la especificación queda definida de una forma mucho más precisa a bajo nivel.

4.3.3 Caso de uso

El listado de la Tabla 11, es un ejemplo completo de un mensaje Biztalk, que servirá para comentar las diferentes partes de que consta.

El formato del mensaje es el típico de SOAP, con una cabecera y un cuerpo dentro de la etiqueta de sobre (*Envelope*).

En la cabecera del mensaje se ponen las diferentes etiquetas típicas de Biztalk (Biztags), siendo las dos que aparecen en el ejemplo las únicas de obligada inclusión (el '`mustUnderstand=1`' implica que de no ser reconocidas por el receptor, el mensaje debe ser descartado):

- *Enpoints*: las dos etiquetas <to> y <from> que anida, se utilizan para indicar mediante otra etiqueta <address>, las entidades de negocio emisora y receptora del mensaje.
- *Properties*: mediante diferentes etiquetas (*identity*, *sendAt*, *expiresAt* y *topic*) se indican las propiedades asociadas al mensaje Biztalk en cuestión.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
    <SOAP-ENV:Header>
      <eps:endpoints SOAP-ENV:mustUnderstand="1"
        xmlns:eps="http://schemas.biztalk.org/btf-2-0/endpoints"
        xmlns:agr="http://www.trading-agreements.org/types/">
          <eps:to>
            <eps:address xsi:type="agr:department">Book Orders</eps:address>
          </eps:to>
          <eps:from>
            <eps:address xsi:type="agr:organization">Book Lovers</eps:address>
          </eps:from>
        </eps:endpoints>
        <prop:properties SOAP-ENV:mustUnderstand="1"
          xmlns:prop="http://schemas.biztalk.org/btf-2-0/properties">
         <prop:identity>uuid:74b9f5d0-33fb-4a81-b02b-5b760641c1d6</prop:identity>
         <prop:sentAt>2000-05-14T03:00:00+08:00</prop:sentAt>
         <prop:expiresAt>2000-05-15T04:00:00+08:00</prop:expiresAt>
         <prop:topic>http://electrocommerce.org/purchase_order/</prop:topic>
        </prop:properties>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
      <po:PurchaseOrder xmlns:po="http://electrocommerce.org/purchase_order/">
        <po:Title>Essential BizTalk</po:Title>
      </po:PurchaseOrder>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Tabla 11. Ejemplo de mensaje Biztalk

Todas las etiquetas Biztags (y por supuesto estas dos que se acaban de ver), están definidas dentro de los espacios de nombres estándares derivadas por extensión del prefijo '*http://schemas.biztalk.org/btf-2-0/*'.

A continuación aparece el cuerpo del mensaje cuyo contenido es un documento Biztalk que a su vez puede contener uno o varios documentos de negocio (por ejemplo una orden de compra, elemento a comprar, dirección donde remitir el envío, etc.).

Otras posibilidades que ofrecen los mensajes Biztalk son:

- *Entrega fiable*: mediante los datos especificados con la etiqueta opcional <service>.
- *Información sobre catálogos*: mediante la etiqueta opcional <manifest>.
- *Gestión de procesos*: mediante la etiqueta opcional <process>.
- Envío de *ficheros adjuntos*: mediante estructuras MIME multi-partes o mediante referencias en la etiqueta <manifest>.
- *Seguridad en documentos y mensajes*: para proporcionar una seguridad con un nivel de granuralidad más fino que el que puede ofrecer SSL (firmar o cifrar parte de un mensaje). Para todo ello se utiliza S/MIME.

4.4 ebXML

4.4.1 Introducción

En las últimas décadas, el intercambio electrónico de datos, EDI (*Electronic Data Interchange*), ha hecho posible que las compañías eliminen la transmisión de documentos escritos, reduciendo los costes e incrementando la eficiencia mediante el intercambio de información de negocio en formato electrónico.

Idealmente, compañías de cualquier tamaño podían utilizar EDI para llevar a cabo transacciones electrónicas, sin que fuese necesario ningún acuerdo previo. En la práctica, sólo las grandes compañías podían permitirse una implementación completa de EDI.

Ya se ha comentado como en los últimos años, XML se ha convertido de una forma muy rápida en el estándar indiscutible de representación e intercambio de datos en Internet para las nuevas aplicaciones de comercio electrónico. Sin embargo, hay grandes inversiones hechas en EDI y además esta tecnología ostenta gran cantidad de experiencia en el sector.

La especificación de la arquitectura ebXML [ebXML], proporciona un marco de desarrollo en el que se puede explotar la nueva funcionalidad que ofrece XML, preservando las inversiones hechas en EDI.

El objetivo de ebXML es promocionar un mercado electrónico global en el que empresas de cualquier tamaño y en cualquier ubicación geográfica puedan encontrarse y llevar a cabo negocios entre ellas mediante el intercambio de mensajes basados en XML.

4.4.2 Arquitectura

Aunque posteriormente se introducirá un ejemplo con un caso de uso, esta sección presenta de forma genérica la arquitectura ebXML, empezando con las características funcionales y siguiendo con las bases sobre las que se asienta toda esa funcionalidad.

Desde un punto de vista funcional, las aportaciones que ofrece ebXML son las siguientes:

- Un mecanismo estándar para describir un proceso de negocio y su modelo de datos asociado.
- Un mecanismo para registrar tanto los procesos de negocio como los meta-modelos de información de forma que puedan ser reutilizados y compartidos con facilidad.
- Un mecanismo para descubrir la información de cada participante:
    - *Procesos de negocio* que soporta.
    - *Interfaces de servicio* que ofrecen para cada proceso de negocio.
    - Los *mensajes de negocio* que se intercambian entre las interfaces de servicio.

  - La configuración de los protocolos de transporte de datos, seguridad y codificación.
- Un mecanismo para registrar la mencionada información de los participantes de forma que pueda ser descubierta y obtenida fácilmente.
- Un mecanismo para describir la ejecución de un acuerdo mutuo entre los participantes de un intercambio de información (*Collaboration Protocol Agreement*, CPA), que pueda deducirse de la información registrada.
- Un entorno estándar de mensajería que permita interoperar de forma segura y fiable.
- Un mecanismo de configuración del servicio de mensajería que permita adecuarlo a los procesos de negocio acordados con las restricciones establecidas.

Aunque el modelado de procesos e información no es obligatorio, si los usuarios o los implementadores deciden modelar dicha información, se recomienda la utilización de una metodología específica basada en UML: UMM (*UN/CEFACT Modeling Methodology*).

Dicha metodología distingue entre dos diferentes aspectos o vistas que definen la arquitectura del sistema: la vista operacional y la vista funcional o de servicio.

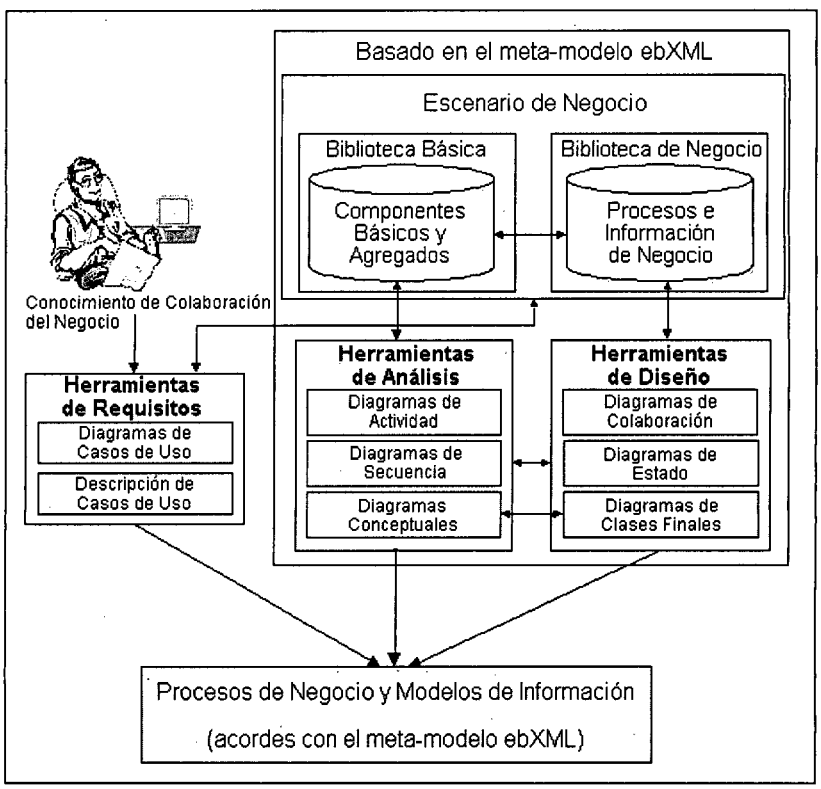


Figura 45. *Business Operational View* del modelo de datos de ebXML [ebXML]

De cara a la descripción de información, el esquema que se utiliza en ebXML es el que ilustra la Figura 45 y se corresponde con lo que se ha denominado 'vista operacional' (*ebXML Business Operational View*, BOV). Más concretamente, esta vista describe la semántica asociada a los procesos de negocio en un formato que es independiente de cualquier lenguaje de programación (UMM). Una vista funcional (*Funtional Service View*, FSV) más específica, se utiliza para modelar la descripción de las funcionalidades

de una entidad, interfaces de servicio, servicios y protocolos asociados a la mensajería que permiten a las entidades encontrarse una a otras e intercambiar sus perfiles.

4.4.3 Caso de uso

Lo primero que debe hace una entidad si quiere pertenecer a un escenario de negocio ebXML y poder interactuar con otras empresas, es comunicar sus características al registro ebXML correspondiente. Para ello tiene dos opciones:

- Utilizar las características de los negocios que actualmente ya están declarados en el registro ebXML por si parte de la información declarada fuese reutilizable (paso 1 en la Figura 46). Para eso debe obtener las especificaciones ebXML que estén en el registro (procesos y escenarios definidos en XML). Dicha información se encuentra tanto en la librería base (*core library*), con contenidos proporcionados por la iniciativa ebXML como en información adicional definida previamente por otras entidades.
- La compañía puede también comprar o desarrollar su propia implementación ebXML específica para sus transacciones y definir nuevos procesos de negocio.

El objetivo es que a base de que los fabricantes vayan proporcionando descripciones, al final el registro contenga prácticamente todos los posibles modelos de negocio y las diferentes entidades no tengan más que obtener de allí lo que necesitan. En el ejemplo de la figura, se supone que la compañía B obtiene del registro todo lo que le hace falta, por lo que no necesita desarrollar una nueva implementación.

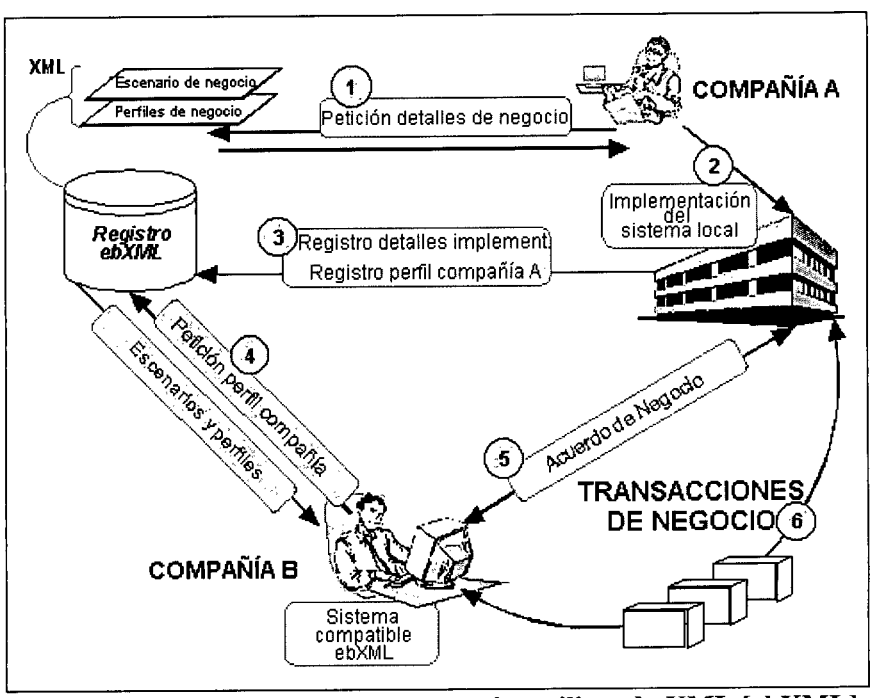


Figura 46. Interacción de dos compañías utilizando XML [ebXML]

Además de almacenar los detalles de la implementación, las compañías deben dar de alta también en el registro sus perfiles (*Collaboration Protocol Profile*, CPP) que especifican entre otras cosas las interfaces de servicio que soporta y varios datos como

protocolos de transporte, seguridad, mensajería, etc. Dichos CPPs vienen nuevamente en forma de documentos XML.

El resto de las compañías pueden obtener estos CPP de cara a conocer cuáles son las características del resto de empresas y determinar si son compatibles con sus requisitos y si es posible llevar a cabo con ellas alguna transacción (paso 4).

De existir dicha compatibilidad, se puede establecer automáticamente un acuerdo de colaboración (*Collaboration Protocol Agreement*, CPA) que ambas entidades intercambian (documentos XML) deducido directamente de los CPPs (paso 5).

Por último, ambas compañías pueden iniciar sus transacciones basadas en mensajes de negocios acordes a estándares de la especificación ebXML.

4.5 eCo

4.5.1 Introducción

Al igual que ocurre con otras arquitecturas de comercio electrónico que se están proponiendo en la actualidad, la arquitectura eCo [eCo] surge principalmente motivada por el objetivo de lograr interoperabilidad en las transacciones comerciales de diferentes empresas.

A lo largo de estos últimos años de desarrollo explosivo de tecnologías relacionadas con el comercio electrónico y de diferentes propuestas de modelos de negocio, lo cierto es que el objetivo de la interoperabilidad se complica cada vez más.

Debido a los numerosos estándares existentes relativos a las diferentes facetas de las transacciones de comercio electrónico, es muy improbable que dos entidades que quieran llevar a cabo negocios electrónicos se basen en sistemas con los mismos protocolos, documentos y modelos transaccionales.

Por eso, el grupo de trabajo de eCo, no se ha centrado en el desarrollo de un estándar para ser utilizado por todas las entidades y de hecho no constituye una alternativa a los actuales protocolos de comercio electrónico. En vez de eso y en la línea de las arquitecturas que acabamos de ver, se ha creado un entorno en el que se puedan publicar y descubrir protocolos existentes, interfaces y tipos de documentos que utilizan los sistemas para procesar una transacción.

La arquitectura eCo parte de la base del conocimiento de otras arquitecturas y de la teoría de que cada una de esas otras arquitecturas, pese a contribuir al desarrollo de una determinada industria, contribuyen también al aumento del problema de la interoperabilidad pues o bien directamente incrementan la variedad de las posibles soluciones, o bien se solapan en determinados puntos.

Así pues, eCo presenta un marco genérico independiente del negocio que se quiera describir o de las entidades que quieran interoperar. eCo busca más la interoperabilidad entre diferentes negocios que la posible estandarización de una arquitectura de comercio electrónico o de un protocolo de comunicación.

Para lograrlo, define una serie de niveles de interoperabilidad de tal forma que dos entidades puedan acordar un contrato para llevar a cabo negocios en común. Una diferencia sustancial con respecto a otras arquitecturas, es que los detalles de la implementación de los negocios no están presentes en los niveles, sino que lo que se definen son meta-datos con los que poder especificar esos detalles. De esta forma, los negocios tienen la libertad de diferenciarse a ellos mismos como prefieran para lograr la interoperabilidad.

Y en vez de buscar interoperabilidad como un objetivo global, la idea en eCo es buscarla siguiendo los niveles que se han comentado. Empezando por descubrir los servicios que ofrece una determinada entidad, el siguiente paso sería evaluar las interfaces de aplicación que ofrece (definidas en documentos genéricos XML). De la misma forma, e incrementando el nivel de interoperabilidad sería interesante construir comunidades o empresas virtuales con los servicios descubiertos que tengan relación entre sí.

El proyecto eCo surge en 1998 bajo el auspicio de CommerceNet y contando con Commerce One como uno de los principales patrocinadores. Actualmente, el proyecto lo desarrolla un grupo de trabajo neutral desde el punto de vista industrial, que incluye miembros de importantes compañías (3Com, American Express, Cisco Systems, Compaq, Hewlett-Packard, IBM, Oasis, Intel, ISO, Microsoft, NEC, Netscape, NIST, Novell, Sun Microsystems, etc.). Además cuenta con expertos involucrados en el desarrollo de otras arquitecturas como OBI, IOTP, RossetaNet, etc.

4.5.2 Arquitectura

eCo proporciona soluciones para que dos entidades heterogéneas puedan establecer una relación B2B. Para ello es necesario que sean capaces de:

- Encontrar otros negocios en Internet.
- Determinar si desean hacer negocios y cómo pueden participar en un mercado.
- Determinar qué servicios son proporcionados y consumidos por otros negocios.
- Determinar las interacciones que tienen lugar y la información y documentos intercambiados en esas interacciones.
- Determinar cómo y cuando pueden comunicarse sus sistemas de comercio electrónico.
- Determinar qué modificaciones hay que implementar, si es necesario, para lograr la interoperabilidad entre los sistemas.
- Si se desea, pueden establecer comunicaciones por otros canales diferentes de Internet.

Las tres principales contribuciones de eCo en el orden de estos requisitos son:

- Un modelo conceptual para describir sistemas de comercio electrónico.
- Un mecanismo para pedir información sobre un sistema de comercio electrónico.
- Definición de documentos para peticiones y respuestas utilizados para el intercambio de información.

Gracias al marco conceptual definido por la arquitectura eCo, las entidades pueden definir y publicar descripciones basadas en meta-datos sobre sus sistemas de comercio electrónico. De esta forma, las entidades interesadas pueden comprender fácilmente los diferentes aspectos del sistema que se representa y proporcionar la información necesaria para fijar los requisitos que se acaban de ver y lograr la interoperabilidad.

4.5.2.1 Modelo conceptual en capas

La primera pieza del entorno especificado por eCo es un modelo conceptual para describir sistemas de comercio electrónico. Dicho modelo está establecido en base a una arquitectura de siete niveles, dedicándose cada uno de ellos a la descripción de una parte diferente del sistema (ver Figura 47).

Las tres primeras capas (*redes*, *mercados* y *negocios*) se centran en el descubrimiento de las ubicaciones que proporcionan un determinado producto o servicio. Las últimas cuatro se centran en la descripción de un mecanismo para conducir una transacción de comercio electrónico que permita la adquisición de un producto o servicio encontrado en algún negocio proporcionado por las tres capas superiores.

- ***Capa de red***: representa redes físicas donde pueden estar ubicados los sistemas de comercio electrónico (Internet por ejemplo). Esta capa proporciona un mecanismo para encontrar mercados que existen en una determinada red.
- ***Capa de mercado***: representa mercados *on-line* que ofrecen productos y servicios. Cada mercado puede tener sus propias reglas, procedimientos y protocolos.
- ***Capa de negocio***: proporciona información sobre un negocio particular (productos que proporciona, ubicación, página Web, contactos, etc.).
- ***Capa de servicio***: esta capa proporciona un listado de todas las interacciones de comercio electrónico o servicios que puede proporcionar un determinado negocio (navegación por catálogos, chequeo de inventario, orden de productos, compra *on-line*, etc.).
- ***Capa de interacciones***: define los pasos necesarios que deben tener lugar entre un sistema de comercio electrónico de un negocio y otro sistema para utilizar un determinado servicio.
- ***Capa de documentos***: esta capa define los tipos de documentos que se utilizarán en los intercambios que generan las interacciones.
- ***Capa de elementos de información***: esta última capa define los datos de que constan los documentos.

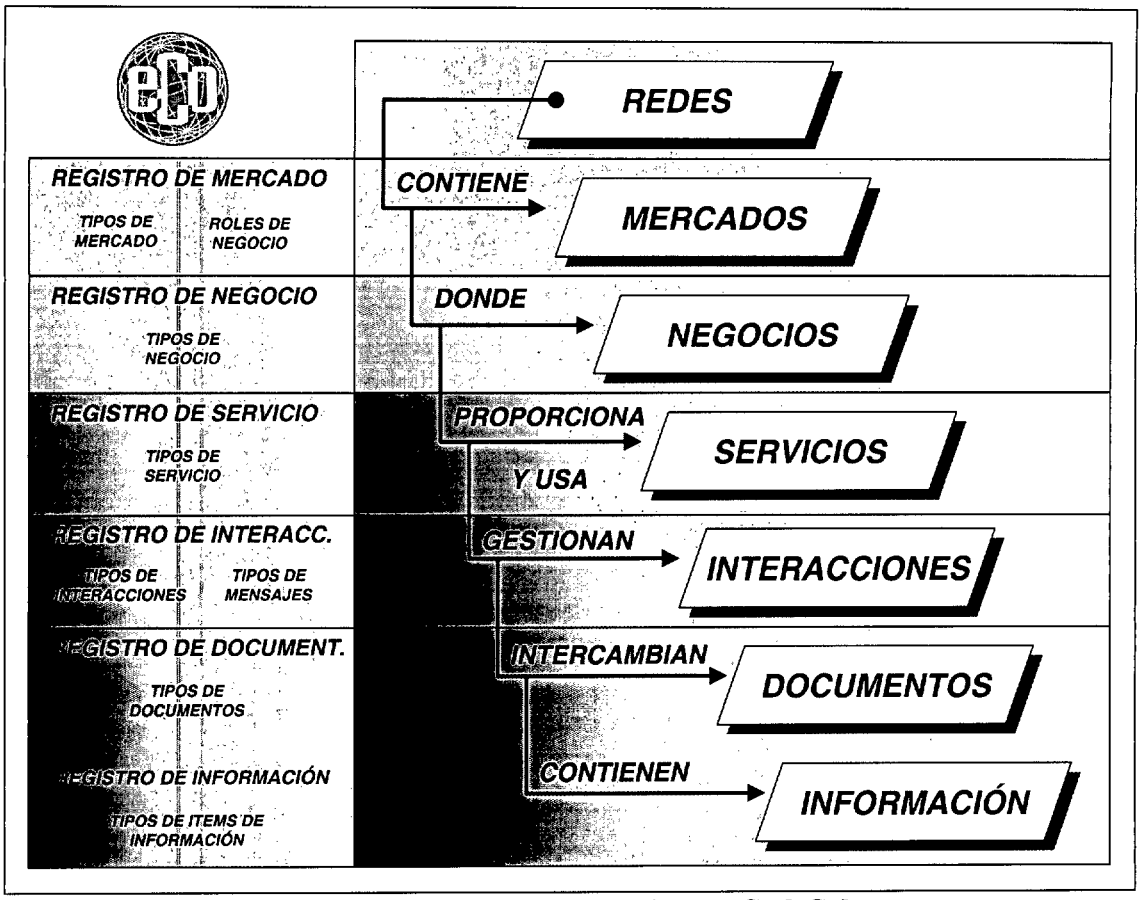


Figura 47. Arquitectura de niveles eCo [eCo]

4.5.2.2 Registros

La información relativa a cada una de las capas arquitecturales (documentos y tipos de componentes) reside en lo que se denomina *registros de tipo* (columna izquierda de la Figura 47). Un registro de la capa de red contendrá una lista de mercados y uno de la capa de mercados contendrá una lista de los negocios que son parte de dicho mercado, etc.

Hay que destacar que los registros no contienen información específica de ninguna implementación o instancia de comercio electrónico. Su objetivo es ser una fuente común de información sobre los posibles tipos referenciados por diferentes implementaciones.

Los registros están estructurados de forma jerárquica, de tal manera que un tipo puede estar definido en función de subtipos (un mercado de componentes de automóviles por ejemplo, puede estar formado por un mercado de neumáticos, motores, etc.).

Cada registro tiene una interfaz gracias a la cual es posible hacer peticiones (definiciones de tipos, etc.). Dichas peticiones se hacen basándose en un protocolo orientado a URLs (típicamente HTTP o HTTPS con el formato típico de las peticiones que salen de formularios HTML) y así se puede reutilizar la infraestructura Web existente para construir y recibir peticiones y respuestas.

4.5.2.3 Definición de peticiones y respuestas

Tanto las peticiones que pueden ejecutarse en una infraestructura eCo, como los documentos que se envían como respuesta, están publicados en la interfaz de cada registro. Las interfaces asociadas a cada uno de los niveles, detalladas por la especificación eCo, proporcionan la siguiente información:

- El nombre de las peticiones que entiende el registro.
- Los argumentos opcionales y obligatorios de cada petición.
- Un DTD (ver 2.2.3.2) con la definición del documento que envía el registro en respuesta a una petición dada.

4.5.3 Caso de uso

En la figura del ejemplo (Figura 48), dos compañías aparecen como 'negocios' dentro de eCo (implementaciones de la capa de negocio de eCo). Dichos negocios coexisten dentro de un mercado eCo, que proporciona un punto de encuentro para el tipo de comercio electrónico que ambas entidades representan. Dicho mercado está presente también dentro de una red de mercados eCo, de forma que puede ser localizado a través de Internet.

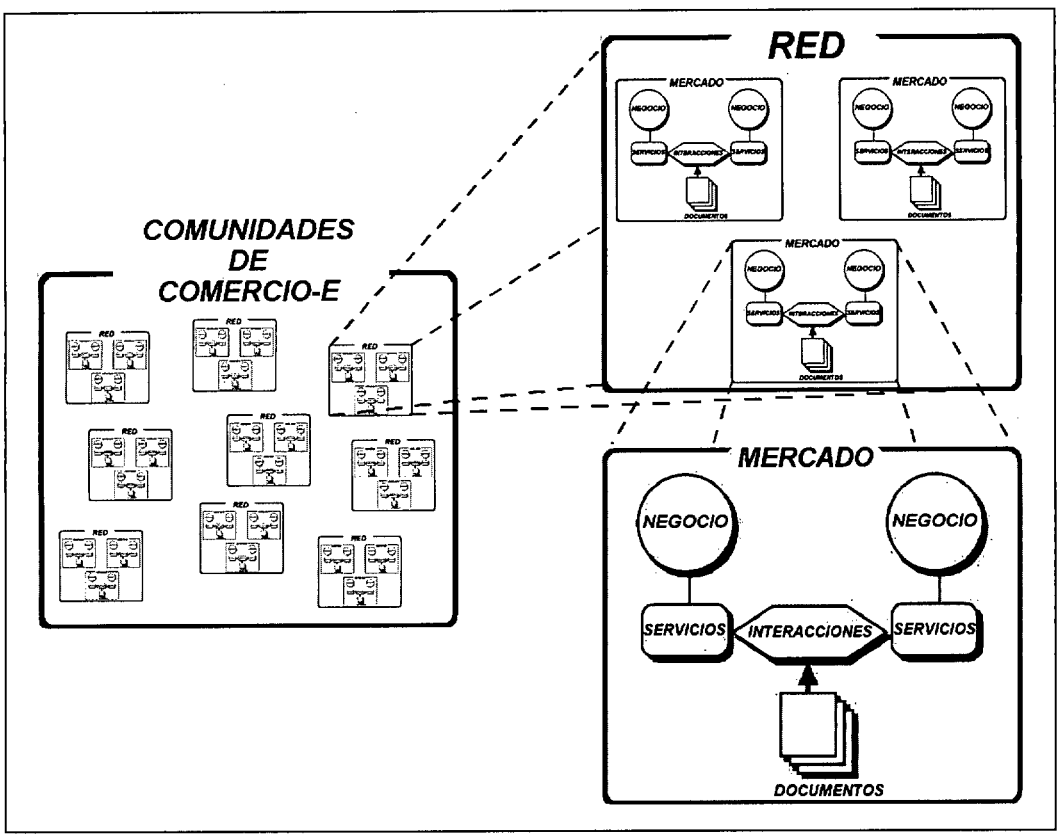


Figura 48. Relaciones de la arquitectura eCo [eCo]

Cada uno de los negocios ofrece una serie de servicios de comercio electrónico (revisar catálogo, ordenar un producto, etc.), que representan interfaces de procesos de negocio. Los negocios interactúan utilizando sus servicios dentro del contexto de algún

proceso de negocio. Los servicios se definen por una serie de intercambios de documentos (dichos intercambios es lo que se denomina interacciones). Una interacción tiene lugar cuando una petición (uno o más documentos) se envía de un negocio a otro y posteriormente se recibe una respuesta (nuevamente uno o más documentos). Y finalmente esos documentos se definen a partir de elementos de información.

Una determinada entidad podría preguntar en la interfaz de una red por los mercados que hay en ella (`http://www.commerce.net/eco/NetworkGetProperties`) y obtendría por respuesta las interfaces de dichos mercados (lo habitual es que dicha red contenga un fichero denominado `eco.xml` en su directorio Web raíz describiendo sus características, por lo que éstas podrían ser localizadas e indexadas automáticamente por un robot). La petición anterior devolvería un fichero XML acorde al siguiente DTD:

```
<elementType name="NetworkPropertySheet">
   <refines><archetypeRef name="eCo" schemaAbbrev="eCo"/></refines>
   <sequence>
       <sequence>
         <elementTypeRef name="Name" minOccur="1" maxOccur="*"/>
         <elementTypeRef name="Maker" minOccur="1" maxOccur="*"/>
         <elementTypeRef name="Operator"/>
       </sequence>
       <sequence>
         <elementTypeRef name="MarketRegistryLocation" minOccur="0" maxOccur="1"/>
         <elementTypeRef name="RegistrationService" minOccur="0" maxOccur="1"/>
         <elementTypeRef name="TermsAndConditions" minOccur="0" maxOccur="1"/>
         <elementTypeRef name="Credentials" minOccur="0" maxOccur="1"/>
       </sequence>
         <elementTypeRef name="NetworkSpecificProperties" minOccur="0" maxOccur="1"/>
    </sequence>
</elementType>
```

Tabla 12. Fichero DTD de ejemplo de mercados eCo

De la misma forma podría preguntar a la interfaz del mercado por los negocios que contiene y a estos por sus servicios e ir concretando más y más hasta llegar a establecer un intercambio de documentos de negocio.

La especificación de eCo detalla las interfaces que pueden accederse en los diferentes registros y niveles de la arquitectura.

Por último, también se indica en la especificación que no todas las interfaces tienen porqué ser implementadas. Únicamente el nivel de negocio es obligatorio. El resto de niveles y registros son opcionales.

4.6 OBI

4.6.1 Introducción

La especificación OBI (*Open Buying on the Internet*, [OBI]) fue creada en 1997 por el consorcio OBI como un entorno de comercio B2B para Internet. En Mayo de 1997 se publicó la versión 1.0 apoyada por American Express y bajo el soporte de SupplyWorks Inc. En 1998 CommerceNet relevó a SupplyWorks en la gestión del proyecto y ese mismo año apareció la versión 1.1 seguida de la 2.0 en 1999. Actualmente ya está

disponible la versión 3.0 y el consorcio cuenta con más de 80 participantes (American Express, Dell Computer, IBM, MasterCard, Microsoft, Netscape, Oracle, Visa, Xerox, etc.).

El objetivo principal de OBI es el de proporcionar un entorno estándar de comercio B2B en Internet interoperable y seguro, presentando un énfasis especial en marcos en los que hay un elevado número de transacciones con intercambios monetarios pequeños.

4.6.2 Arquitectura

La idea de la arquitectura OBI es dejar que las organizaciones vendedoras mantengan sus propios catálogos permitiendo que sean ellas las responsables de diferenciar sus productos y precios. Esto permite que las empresas compradoras no tengan que mantener ningún tipo de catálogo o software específico para acceder a los mismos.

La arquitectura OBI consta de cuatro componentes fundamentales:

- ***Comprador (requisitioner)***: es un miembro de la empresa compradora que utiliza un navegador para acceder al Web de su propia empresa y desde allí a los catálogos de las organizaciones vendedoras (proveedores). Es la persona que realmente ordena los productos o servicios y se supone que tiene acceso a Internet desde un navegador Web así como un certificado digital expedido por una autoridad de certificación.
- ***Empresa compradora***: se encarga de mantener los perfiles de los usuarios, las listas de los proveedores y proporciona el proceso de confirmación de órdenes de compras. Esta organización debe tener un servidor OBI para recibir peticiones y emitir órdenes. La organización compradora también mantiene relaciones contractuales con sus preferidas empresas vendedoras.
- ***Empresa vendedora***: se encarga de autentificar usuarios, proporcionar vistas de catálogos y precios y de procesar órdenes de compra. Mantiene catálogos dinámicos con información precisa sobre productos y sus precios. Dicha información está perfectamente integrada con un servidor OBI para recibir órdenes y emitir peticiones.
- ***Autoridad de pago***: se encarga de facturar y recibir pagos de acuerdo al formato de pago elegido por el usuario.

La especificación técnica se centra en los siguientes aspectos: el proceso estándar gracias al cual un comprador accede a un catálogo especializado, un formato estándar para intercambiar órdenes de compra y mecanismos estándares de seguridad.

Con respecto a la tecnología subyacente, la arquitectura OBI pretende utilizar estándares existentes de cara a maximizar la interoperabilidad y a minimizar los costes de implementación: el intercambio de documentos se basa en EDI (concretamente X12 850 EDI) y en EDIFACT, el transporte en HTTP, la seguridad en SSL, la firma digital en PKCS#7 y los certificados en X.509. La especificación actual está adaptando el formato de algunos documentos a XML.

4.6.3 Caso de uso

La siguiente lista describe la secuencia de acciones que tienen lugar desde el momento en que un comprador accede al sistema hasta que obtiene la confirmación de la comprar por parte de la autoridad de pago:

1. El comprador mediante un navegador accede a su servidor OBI ubicado en la organización compradora y selecciona el Web de alguna organización vendedora (proveedores) con catálogos *on-line*, etc.
2. Automáticamente se le pone en contacto con la organización vendedora, que lo autentica a él y a su organización y gracias a los perfiles de usuario correspondientes, le muestra una vista personalizada de los catálogos. El usuario navega por ellos y selecciona los que considera oportunos.
3. El carro de la compra del usuario se convierte en una petición (compatible con EDI) que se firma y se encapsula en un objeto OBI, se codifica y se envía de forma segura a la organización compradora usando HTTP y SSL, bien mediante el método servidor-servidor (3) o el servidor-navegador-servidor (3a-3b). La organización compradora obtiene el mensaje y tras desencapsularlo lo convierte a un formato interno.
4. Se le añade información administrativa y de pago (que puede obtenerse automáticamente del perfil) y la orden se procesa internamente.
5. La orden se firma, se encapsula en un mensaje OBI y se transmite de forma segura a la organización vendedora que lo traduce a un formato interno.
6. Opcionalmente se puede contestar con información sobre cómo se formalizará la orden.
7. También opcionalmente la organización compradora puede incluir detalles adicionales sobre la orden.
8. La organización vendedora obtiene crédito si es necesario e inicia el proceso de venta.
9. La autoridad de pago extiende una factura y recibe el pago de la transacción.

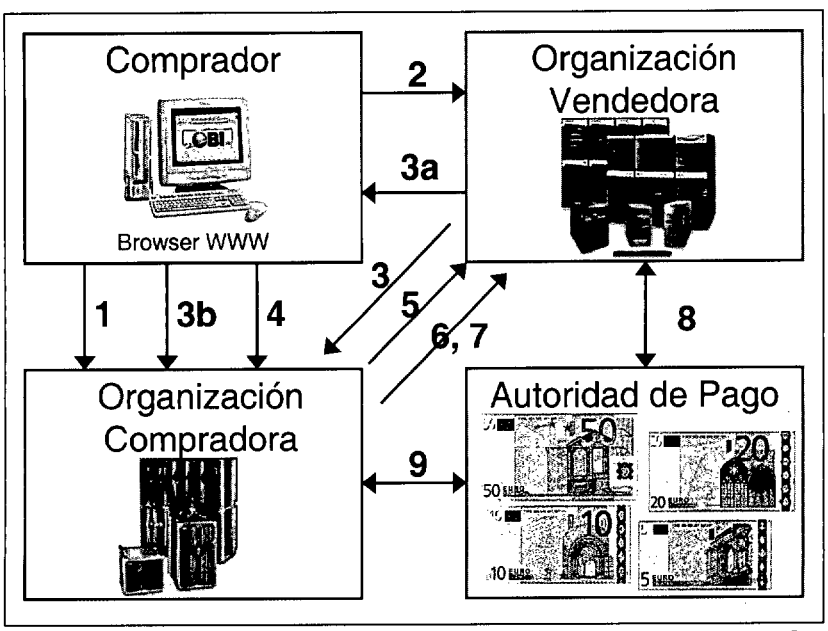


Figura 49. Secuencia de transacciones que tienen lugar en OBI [OBI]

4.7 Otras arquitecturas

4.7.1 RosettaNet

RossetaNet [Rosetta] es un consorcio sin ánimo de lucro fundado en 1998 con 40 empresas del sector de las tecnologías de la información que consta actualmente con más de 400 empresas líderes del sector de las tecnologías de la información, componentes electrónicos, fabricantes de semiconductores y proveedores de servicios, con el objetivo común de crear, implementar y promocionar estándares abiertos de procesos de negocio electrónico.

El consorcio, pretende diferenciarse de otras iniciativas similares gracias a un proceso rápido de creación de estándares al que contribuyen activamente gran cantidad de expertos de diferentes sectores.

Los estándares de RossetaNet disponibles de forma gratuita en el Web de la organización, incluyen:

- ***Diccionario de negocio***: especifica las propiedades para definir transacciones de negocio entre las diversas entidades.
- ***Diccionario técnico***: especifica las propiedades para definir productos y servicios.
- ***Entorno de implementación*** (*RossetaNet Implementation Framework*, RNIF): proporciona protocolos de intercambio para permitir una implementación rápida y eficiente de los estándares de RossetaNet. El RNIF especifica los intercambios de información basándose en XML y cubriendo tanto el transporte, el encaminamiento, seguridad, señalización y acuerdo entre entidades.
- ***Interfaces de procesos*** (*Partner Interface Process*, PIP): son diálogos extremo a extremo basados en XML que definen procesos de negocio. Cada PIP incluye un documento de negocio con el vocabulario, el proceso de negocio y la *coreografía* del intercambio de mensajes. Los PIP están clasificados dentro de siete grupos (clusters) de procesos de negocios básicos que representan el núcleo de una red de negocio. Cada grupo se divide en segmentos representando procesos entre empresas y dentro de esos segmentos están ubicados los PIPs individuales.

4.7.2 Building Blocks

El proyecto fue desarrollado bajo el patrocinio de CommerceNet y se centró fundamentalmente en las siguientes actividades:

- Identificar procesos de negocio para el comercio electrónico.
- Identificar funciones técnicas para apoyar los procesos de negocio.
- Identificación de las implicaciones técnicas más relevantes de los detalles legales y reguladores del comercio electrónico.
- Discusión de las facetas de interoperabilidad y estandarización.

El informe identifica los siguientes bloques principales:

- *Procesos de marketing*: consulta y construcción de catálogos de productos, extracción, análisis y publicación de la información, obtención de precios, implementación de cesta de la compra.
- *Procesos de contratación*: petición/confirmación de órdenes, gestión de niveles de stock, cancelación de órdenes.
- *Procesos logísticos*: notificación de entrega/recepción.
- *Procesos de contratación*: selección de los mecanismos de pago y confirmación disponibles, acceso a tarjetas crédito o monederos.
- *Interfaz con procesos de administración*: declaración de impuestos, notificación de exportaciones, etc.

La metodología Building Blocks se ha convertido en una referencia para gran cantidad de arquitecturas de comercio electrónico que se desarrollaron posteriormente y además sirvió para identificar diferentes requisitos de estandarización en el comercio electrónico.

4.7.3 OTP

IOTP (*Internet Open Trading Protocol*, [Bur00]), fue originalmente desarrollado por el consorcio OTP y tras su desaparición pasó a ser gestionado por un grupo específico del IETF.

Los objetivos principales de IOTP son:

- Permitir el desarrollo de productos interoperables para el soporte del comercio electrónico (usuarios compatibles con IOTP pueden llevar a cabo operaciones con comerciantes compatibles con IOTP).
- Replicar las experiencias comerciales del mundo real en el mundo virtual.
- Proporcionar una experiencia de 'tienda universal' (interfaz consistente para todos los pasos de la negociación independientemente de la identidad de las entidades).
- Encapsular cualquier mecanismo de pago existente en Internet.
- Responder a la demanda del mercado (empezar con protocolos básicos simples e ir añadiendo protocolos adicionales a medida que el mercado los requiera).

Se identifican cuatro tipos de intercambios de negocio:

- *Ofertas*: el comerciante le ofrece al consumidor un motivo por el cual llevar a cabo una operación comercial. El usuario debe aceptar la oferta para la que la operación continúe.
- *Pagos*: el pago implica algún tipo de intercambio monetario entre el consumidor y la entidad de pago.
- *Distribución*: entrega efectiva, bien sea on-line o física, de los productos al consumidor.
- *Autenticación*: puede ser utilizado por cualquier entidad que tome partido en algún proceso de negocio para exigir la identidad de otra entidad y asegurarse de que es quien dice ser.

La especificación IOTP, que está basada en XML, fija los contenidos, el formato y la secuencia de mensajes que se intercambian las distintas entidades de los procesos.

4.7.4 Java EC Framework

El origen del entorno de comercio electrónico de Java se encuentra en el desarrollo por parte de Sun Microsystems en 1995, de una serie de APIs para resolver la gran cantidad de problemas que se presentaban al implementar soluciones para el comercio electrónico. En 1996 Sun decidió comercializar la tecnología. El resultado es un conjunto de herramientas y tecnologías construidas sobre las JDK y empaquetadas con el nombre de Java Commerce que comúnmente se conoce como el entorno Java EC. Dichas herramientas proporcionan una infraestructura completa para desarrollar aplicaciones de comercio electrónico basadas en una arquitectura de componentes.

Los criterios de diseño de Java Commerce son:

- Crear un entorno software flexible y seguro para compras, banca y finanzas electrónicas que se ejecute en cualquier hardware, desde tarjetas inteligentes a grandes estaciones de trabajo.
- Crear una plataforma abierta que sea capaz de soportar todo tipo de estándares y protocolos de pago ejecutándose concurrentemente en el mismo entorno independientemente de que este sea un sistema intranet propietario o en Internet.
- Herramientas para el desarrollo que permitan reducir costes, esfuerzo y tiempo en la implementación de soluciones.
- Hacer sencilla la creación de aplicaciones descargables, para cobrar por la información y los contenidos obtenidos.

El entorno se basa fundamentalmente en tres tecnologías: *Java Ballet*, *Commerce JavaBeans* y *Java Card.*

La primera pretende ser una arquitectura independiente de la plataforma para el lado del cliente, que permita juntar de forma transparente diferentes componentes comerciales (protocolos y tecnologías) y proporcionar seguridad a las transacciones comerciales.

Las Commerce JavaBeans son módulos de código Java, reutilizables que implementan protocolos transaccionales específicos (pago con tarjeta, cheques electrónicos, etc.). El Java Wallet está diseñado para obtener directamente por la red *cassettes* (contenedores firmados digitalmente para los componentes Commerce JavaBeans) que son específicas de un tipo particular de transacción y que amplían las posibilidades de la plataforma.

Por último, la especificación Java Card permite la ejecución de tecnología Java en tarjetas inteligentes, así como en otros dispositivos de memoria limitada.

Pese a que Sun ha abandonado el soporte a esta arquitectura sigue apoyando muy activamente el desarrollo de la tecnología Java Card.

4.8 Conclusión

La tremenda acogida que han experimentado los denominados servicios Web, incluso antes de que haya tecnología suficientemente madura como para poder desarrollarlos y mantenerlos, ha hecho pensar que el objetivo final de un mercado global no solo de cara a los usuarios finales, sino también entre empresas, puede ser alcanzable.

Sin embargo, la tecnología empleada para la descripción, el registro y la localización de servidores Web no es capaz por sí sola de cubrir todas las necesidades que requiere un intercambio del estilo del que demanda el comercio B2B.

Por eso se han desarrollado nuevas arquitecturas que definen las diferentes entidades que formarán parte del intercambio electrónico, así como diferentes mecanismos para especificar, registrar, localizar e intercambiar de forma precisa los diferentes procesos de negocio que se pretenden llevar a cabo entre las empresas participantes.

Las diferencias entre ellos quedan marcadas por diferentes aspectos como la utilización de mecanismos de comunicación síncronos o asíncronos, la capacidad de ampliación a otros entornos, los tipos de datos que se utilizan, la posibilidad de transmitir datos binarios, etc. (puede encontrarse una comparativa de las diferentes alternativas en [Pru01]).

Independientemente de las características que tengan las arquitecturas, lo que sí parece que ha evidenciado notables diferencias en cuanto a las posibilidades de implantación de unas y otras es el hecho de que estén o no basadas en tecnologías XML, ya que las líneas de investigación actuales en cuestión de interoperabilidad de aplicaciones, vienen bajo la pauta de la creación de protocolos basados en este lenguaje.

Precisamente por eso, es por lo que las dos arquitecturas que están recibiendo mayor apoyo son Biztalk y sobre todo ebXML (la primera es de Microsoft y la segunda es un estándar abierto). Ambas tecnologías no son sino extensiones al protocolo SOAP (Biztalk incluye de hecho determinadas extensiones que en principio están fuera de las líneas que marca en estándar) junto con una serie de mecanismos para registrar procesos de negocio que puede ser definidos a la medida de las necesidades.

Ambas tecnologías se ven ya como la extensión natural a los servicios Web para proporcionar interoperabilidad entre empresas, puesto que son perfectamente compatibles con las tecnologías que definen los mencionados servicios y además vienen avaladas por el proclamado soporte por parte de los servidores de aplicaciones (.NET evidentemente soportará Biztalk mientras que J2EE dará soporte a ebXML).