PLATAFORMA DE COMERCIO ELECTRÓNICO BASADA EN LA TECNOLOGÍA JXTA

Departamento de Ingeniería de Sistemas y Automática
Área de Ingeniería Telemática
Escuela Superior de Ingenieros, Universidad de Sevilla

Sevilla, Mayo 2005

PROYECTO FIN DE CARRERA
REALIZADO POR **TRINITARIO MESEGUER CALERO**
DIRIGIDO POR **D. ANTONIO SIERRA COLLADO**
DÑA. TERESA ARIZA GÓMEZ

A mis padres, porque a ellos les debo todo lo que soy. Gracias por vuestra paciencia y vuestro constante ánimo.

INDICE

OBJETIVOS DEL PROYECTO 10
1. INTRODUCCIÓN. 12
2. COMERCIO ELECTRÓNICO 15
2.1. INTRODUCCIÓN 15
2.2. CATEGORÍAS DE LAS TRANSACCIONES EN EL COMERCIO ELECTRÓNICO 18
2.3. EL COMERCIO ELECTRÓNICO EN ESPAÑA 20
2.3.1. Introducción 20
2.3.2. El comercio electrónico en cifras 22
2.3.3. Consideraciones Generales 27
2.4. VENTAJAS DEL COMERCIO ELECTRÓNICO DEL FUTURO 29
2.5. ESTÁNDARES DE COMERCIO ELECTRÓNICO 30
2.5.1. EDI 31
2.5.2. EDI-XML 33
2.5.3. BizTalk 33
2.5.4. RosettaNet 34
2.5.5. xCBL 35
2.5.6. UBL 36
2.5.7. ebXML 36
2.5.8. cXML 37
3. PEER-TO-PEER 40
3.1. QUÉ ES PEER-TO-PEER 40
3.1.1. Definición 40
3.1.2. Requisitos de un sistema P2P 40
3.1.3. Lo que no es P2P 41
3.1.4. Características de una red P2P 42
3.1.5. Ventajas del P2P 42
3.2. MODELOS P2P 43
3.3. P2P VS CLIENTE-SERVIDOR 44
3.4. TIPOS DE REDES P2P 45
3.4.1. Sistemas P2P no estructurados 45
3.4.2. Sistemas P2P estructurados 46

3.5. CAMPOS DE APLICACIÓN DEL P2P 47
3.6. SEGURIDAD EN P2P 48
3.6.1. Amenazas externas 49
3.6.2. Amenazas internas 50
3.6.3. El futuro de la seguridad en P2P 51
3.7. RETOS Y FUTURO DEL P2P 52
3.8. REDES P2P 54
3.8.1. Red Freenet. 54
3.8.1.1. Introduccion. 54
3.8.1.2. El problema del anonimato. 54
Anonimato del cliente. 54
Anonimato del servidor. 55
Problemas de seguridad. 55
Soluciones a estos problemas. 56
3.8.1.3. Funcionamiento de Freenet. 56
Freenet es anónima tanto para cliente como para servidor. 56
Freenet consigue que la información más valiosa sea más accesible. 58
Freenet hace que la información menos valiosa desaparezca. 59
Los problemas de Freenet. 59
Freenet en Internet. 62
3.8.2. Red GNUtella. 63
3.8.2.1. Introducción 63
3.8.2.2. Sistemas Distribuidos Emergentes para Compartir Archivos. 64
3.8.2.3. Características de Gnutella 64
3.8.2.4. Estructura de Mensajes 66
1. Función Ping ó Init (código 0x00) 66
2. Función Pong ó Init_Response (código 0x01) 67
3. Función Push (código 0x40) 68
4. Función Query (código 0x80) 69
5. Función Query_Hit (código 0x81) 70
3.8.2.5. Reglas de Propagación 73
3.8.2.6. Descarga de Archivos 75
3.8.2.7. Funcionamiento de una Red Gnutella 76
3.8.2.8. Conclusiones 83
4. TECNOLOGÍAS XML 84
4.1. XML 84
4.1.1. Introducción 84
4.1.1.1. Qué es XML 84
4.1.1.2. Objetivos de XML 85
4.1.2. Aspectos básicos 85
4.1.2.1. Tipos de documentos 85
4.1.2.2. Metalenguajes para definir XML 86
4.1.2.3. Procesado de XML 87
4.1.3. Sintaxis básica 88
4.1.3.1. Estructura general 88
4.1.3.2. Sintaxis de los elementos 89
4.1.3.3. Prólogo y Declaración de un documento 89
4.1.3.4. Comentarios 90
4.1.3.5. Declaraciones de Elementos 90
4.1.3.6. Definiciones de Atributos 90
4.1.3.7. Entidades 91
4.1.3.8. Instrucciones de Procesado 91
4.1.3.9. Secciones CDATA 91
4.1.4. Tecnologías XML 91
4.2. DOM 92
4.2.1. Qué es DOM 92

4.2.2. Aspectos básicos de DOM 93
4.2.3. Niveles de DOM 93
4.3. XSL 95
4.3.1. XSLT 95
4.3.1.1. Qué es XSLT 95
4.3.1.2. Ventajas y desventajas 96
4.3.1.3. Formas de uso 97
4.3.1.4. La transformación XSLT 97
4.3.1.5. Sintaxis XSLT 99
4.3.1.6. Herramientas XSLT 99
4.3.2. XSL-FO 100
4.3.3. XPath 101
4.3.3.1. Objetivos de XPath 101
4.3.3.2. Aspectos básicos 102
4.3.3.3. Algunos ejemplos básicos 102
4.4. XML Y JAVA 104
4.5. JAXP 105
4.5.1. SAX 106
4.5.2. DOM 108
4.5.3. XSLT 109
4.6. JDOM 110
4.6.1. Qué es JDOM 110
4.6.2. Aspectos básicos de JDOM 111
4.6.3. Los paquetes JDOM 112
5. CXML 114
5.1. INTRODUCCIÓN 115
5.1.1. Qué es cXML 115
5.1.2. Capacidades principales de cXML 116
5.1.2.1. Catálogos 117
5.1.2.2. Órdenes de compra 117
5.1.3. Aplicaciones que usan cXML 118
5.1.4. Validación XML 118
5.2. ASPECTOS BÁSICOS DE CXML 120
5.2.1. Elementos y atributos 120
5.2.2. Especificación de protocolo 121
5.2.2.1. Modelo Petición-Respuesta 121
El sobre principal 123
La cabecera de la petición cXML 125
El cuerpo de la petición cXML 126
El cuerpo de la respuesta cXML 126
5.2.2.2. Modelo sentido único (asíncrono) 127
5.3. TRANSACCIONES CXML 128
5.3.1. Transacción de Perfil 128
5.3.1.1. Elemento ProfileRequest 129
5.3.1.2. Elemento ProfileResponse 129
5.3.2. Órdenes de compra 129
5.3.2.1. Elemento OrderRequest 130
5.3.2.2. Respuesta a una OrderRequest 132
5.3.3. Catálogos 133
5.3.3.1. Elementos básicos 133
5.3.3.2. Gestión de suscripciones 134
5.3.4. Otras transacciones 135
6. JXTA 137

6.1. INTRODUCCIÓN 137
6.1.1. ¿Por qué el Proyecto JXTA? 137
6.1.2. ¿Qué es el Proyecto JXTA? 138
6.1.3. ¿Qué puede hacerse con la Tecnología JXTA? 139
6.1.4. Historia de JXTA 139
6.2. LA ARQUITECTURA DE JXTA 140
6.2.1. Visión General 140
6.2.2. Componentes de JXTA 141
6.2.2.1. Aspectos importantes de la arquitectura de JXTA 142
6.3. CONCEPTOS JXTA 142
6.3.1. Peers 142
6.3.2. Grupos de pares 143
6.3.3. Servicios de red 145
6.3.4. Módulos 146
6.3.4.1. Module Class: 147
6.3.4.2. Module Specification: 147
6.3.4.3. Module Implementation: 148
6.3.5. Pipes 148
6.3.6. Mensajes 150
6.3.7. Anuncios (Advertisements) 151
6.3.8. Seguridad 153
6.3.9. Identificadores 155
6.4. ARQUITECTURA DE RED. 155
6.4.1. Organización de la Red 155
6.4.1.1. Par de borde mínimo 156
6.4.1.2. Par de borde de amplio espectro. 156
6.4.1.3. Par Rendezvous (par de citas). 156
6.4.1.4. Relay peer (par de retransmisiones). 157
6.4.2. Índice Distribuido de Recursos Compartidos. 157
6.4.2.1. Consultas 158
6.4.3. Firewalls y NAT's. 159
6.5. PROTOCOLOS JXTA. 160
6.5.1. Peer Discovery Protocol (PDP). 162
6.5.2. Peer Information Protocol (PIP). 163
6.5.3. Peer Resolver Protocol (PRP). 163
6.5.4. Pipe Binding Protocol (PBP). 165
6.5.5. Endpoint Routing Protocol (ERP) 165
6.5.6. Rendezvous Protocol (RVP). 166
6.6. JXTA EN JAVA 167
7. APLICACIÓN DESARROLLADA. 169
7.1. INTRODUCCIÓN. 169
7.2. ANALISIS DE LAS CLASES DESARROLLADAS 169
7.2.1. Clase Comercio 169
7.2.2. Clase PlataformaJXTA 171
7.2.3. Clase Anunciar 172
7.2.4. Clase ServidorServiciosJXTA 172
7.2.5. Clase Vender 173
7.2.6. Clase Buscar 174
7.2.7. Clase ClienteServiciosJXTA 174
7.2.8. Clase TuberiaCliente 175
7.2.9. Clase Comprar 175

7.2.10. Clase DatosDelComprador 176
7.2.11. Clase Categorías 177
7.2.12. Clase ModeloTabla 178
7.2.13. Clase ModeloTablaAnunciados 179
7.2.14. Clase ModeloTablaEncontrados 179
7.2.15. Clase ModeloTablaVendidos 180
7.2.16. Clase ModeloTablaComprados 180
7.2.17. Clase MuestraProducto 181
7.2.18. Clase MuestraProductoEncontrado 181
7.2.19. Clase DocumentoXML 181
7.2.20. Clase ProductoXML 182
7.2.21. Clase DocumentoCXML 184
7.2.22. Clase PeticionCXML 184
7.2.23. Clase PeticionDeCompraCXML 184
7.2.24. Clase PeticionDePerfilCXML 185
7.2.25. Clase RespuestaCXML 185
7.2.26. Clase FactoriaAleatoria 185
7.2.27. Clase FechaISO8601 185
7.2.28. Clase UtilesDirectorio 185
7.2.29. Clase Fichero 186
7.3. FUNCIONALIDADES DE LA APLICACIÓN 186
7.3.1. Anunciar un producto. 187
7.3.2. Buscar un producto. 191
7.3.3. Comprar un producto. 193
7.3.4. Vender un producto. 196
7.3.5. Otras funcionalidades. 197
8. MANUAL DE INSTALACIÓN 199
8.1. INSTALACIÓN DE JAVA 199
8.2. INSTALACIÓN DE "COMERCIO ELECTRONICO 1.0" 199
8.3. EJECUCIÓN DE "COMERCIO ELECTRONICO 1.0" 200
8.3.1. Configuración de JXTA 201
9. PLANIFICACIÓN TEMPORAL DEL PROYECTO 205
10. PRESUPUESTOS 207
10.1. COSTE DE RECURSOS HUMANOS 207
10.2. COSTE DE MATERIAL 208
10.2.1. Coste del software. 208
10.2.2. Coste del hardware. 208
10.2.3. Coste de consumibles. 209
10.3. RESUMEN DEL PRESUPUESTO 209
11. LINEAS FUTURAS. 210
11.1. CENTRALIZACIÓN 210
11.2. SEGURIDAD 211
11.3. MAYOR ESPECIALIZACIÓN 212
11.4. MEJORAS TECNOLÓGICAS 213
12. CONCLUSIONES 215
13. BIBLIOGRAFÍA 217
13.1. ARQUITECTURA "PEER TO PEER": 217
13.2. COMERCIO ELECTRÓNICO 218
13.3. JAVA 219
13.4. JXTA 219

13.5. XML....220
13.5.1. XML, XSL y DOM:....220
13.5.2. JDOM....221
13.5.3. cXML....222
14. ANEXO: CÓDIGO FUENTE DE LA APLICACIÓN....223
1.1. PAQUETE PFC.CODIGO.ACCIONES....223
1.1.1. Clase Anunciar....223
1.1.2. Clase Buscar....226
1.1.3. Clase Comprar....227
1.1.4. Clase Vender....229
1.2. PAQUETE PFC.CODIGO.XML....231
1.2.1. Clase DatosDelComprador....231
1.2.2. Clase Categorías....239
1.2.3. Clase DocumentoXML....241
1.2.4. Clase ProductoXML....248
1.2.5. Clase DocumentoCXML....252
1.2.6. Clase PeticionCXML....255
1.2.7. Clase PeticionDeCompraCXML....263
1.2.8. Clase PeticionDePerfilCXML....270
1.2.9. Clase RespuestaCXML....270
1.3. PAQUETE PFC.CODIGO.JXTA....275
1.3.1. Clase PlataformaJXTA....275
1.3.2. Clase ServidorServiciosJXTA....282
1.3.3. Clase TuberiaCliente....289
1.3.4. Clase ClienteServiciosJXTA....293
1.4. PAQUETE PFC.CODIGO.TABLAS....297
1.4.1. Clase ModeloTabla....297
1.4.2. Clase ModeloTablaAnunciados....301
1.4.3. Clase ModeloTablaEncontrados....303
1.4.4. Clase ModeloTablaVendidos....305
1.4.5. Clase ModeloTablaComprados....306
1.4.6. Clase MuestraProducto....308
1.4.7. Clase MuestraProductoEncontrado....310
1.5. PAQUETE PFC.CODIGO.UTILES....312
1.5.1. Clase FactoriaAleatoria....312
1.5.2. Clase FechaISO8601....313
1.5.3. Clase UtilesDirectorio....318
1.5.4. Clase Fichero....319
1.6. PAQUETE PFC.CODIGO....321
1.6.1. Clase Comercio....321

OBJETIVOS DEL PROYECTO

El objetivo fundamental de este proyecto es el desarrollo de un **sistema de comercio electrónico** basado en dos pilares fundamentales:

- Implementado sobre arquitectura *peer-to-peer* mediante JXTA.

- Uso del estándar de comercio electrónico cXML.

De esta forma, se persigue la total integración de sistemas, en principio muy distintos, bajo el objetivo común de proporcionar un marco fiable y eficiente de comercio electrónico.

En base a este objetivo fundamental, se establecen otros dos objetivos cuyos propósitos complementan y justifican al primero:

- Realizar un estudio de la tecnología *peer-to-peer* y de los diferentes tipos de redes que la implantan, en particular el conjunto de protocolos JXTA, con clara orientación práctica, de cara al desarrollo de aplicaciones sobre esta plataforma. La investigación debe apuntar hacia la obtención de conclusiones referentes a la posible implantación de sistemas de comercio electrónico sobre modelos *peer-to-peer*.

- Realizar un estudio de la situación actual del comercio electrónico, principalmente en nuestro país, haciendo especial hincapié en los estándares y lenguajes orientados al comercio desarrollados con la tecnología XML.

La arquitectura P2P constituye también en sí misma un nuevo modelo de negocio entre empresas. Los anteriores modelos de negocio se basan en arquitecturas cliente-servidor, en la que una empresa o un actor del modelo de negocio actúa con un papel preponderante sobre los demás, conduciendo las transacciones entre los participantes. Pero a veces esto no es útil ni real, puesto que muchas relaciones comerciales son entre iguales, y la presencia de intermediarios ajenos al proceso hace que se pierda el concepto de igualdad entre los participantes, estando toda la transacción supeditada a la organización tercera.

Sin embargo, en el modelo de negocio P2P, las relaciones comerciales son entre iguales: las transacciones comerciales entre empresas (B2B) ya no estarán controladas por una entidad externa, o se puede controlar su papel. En P2P, todos los miembros son iguales y nadie tiene, en principio, un papel preponderante sobre los demás: las relaciones B2B encontrarían aquí su marco ideal de desarrollo. Esto no quiere decir que no pueda haber intermediarios en las transacciones: el modelo P2P es lo suficientemente abierto como para admitir un participante con un papel especial, incluso la ejecución de transacciones mediante un servidor centralizado al que la red P2P está conectado. Las posibilidades del P2P aquí son enormes.

La unión del modelo de negocio P2P con las tecnologías de repositorios XML permitirán relaciones comerciales igualitarias entre empresas iguales previo acuerdo de la forma en la que se negociará gracias a los objetos de negocio guardados en dichos repositorios XML.

1. INTRODUCCIÓN.

El gran crecimiento de Internet en los últimos años ha motivado el desarrollo de numerosas iniciativas de cara a la implantación de sistemas de comercio electrónico sobre la red. Se han ido abandonando los sistemas tradicionales de intercambio de datos electrónicos en favor de nuevas aplicaciones que usan tecnologías mucho más flexibles, baratas y eficientes, como son las tecnologías XML. Se prevé que el comercio electrónico, tanto entre empresas como los usuarios, ocupe en breve gran parte del tráfico en la red.

Asímismo, la tecnología *peer-to-peer* se ha divulgado ampliamente, a veces por motivos ajenos a lo tecnológico, y está siendo reconocida como un modelo de red con un futuro muy prometedor. Incluso se está considerando la arquitectura *peer-to-peer* como un modelo de negocio. Sin embargo, existe aún gran recelo sobre el empleo de esta tecnología, de nuevo motivado por asuntos poco relacionados con los tecnológicos.

Sería interesante estudiar qué resultado se obtendría de la integración de estas nuevas tecnologías, en especial de la combinación de la escalabilidad y potencia del modelo *peer-to-peer* con los modelos de negocio basados en XML. El desarrollo de tal sistema supondría un nuevo modelo de comercio electrónico, en el que las relaciones son de igual a igual y no existen, en principio, terceros que actúan como intermediarios que conducen las transacciones.

La presente memoria se divide en 12 capítulos cuyo contenido se resume a continuación:

1. ***"Introducción"***: Se trata del presente capítulo en el que se trata de introducir este Proyecto Fin de Carrera. Se trata de dar una visión general de lo que se pretende desarrollar.

2. ***"Comercio Electrónico"***: Con este apartado se pretende dar unas ideas básicas sobre el comercio electrónico, el estado actual de éste en España y las principales tecnologías existentes actualmente al respecto.

3. ***"Peer-to-Peer"***: Capítulo dedicado a dar una visión general de las redes P2P, los diferentes tipos de redes de este tipo que existen y se analiza en profundidad dos de ellas: la red

Freenet y la red Gnutella

4. ***"Tecnologías XML"***: En este apartado al igual que en los dos siguientes se pretende entrar con una relativa profundidad en las distintas tecnologías por las que se ha optado al desarrollar este proyecto. Se trata de entrar en detalle en el lenguaje XML y en las tecnologías relacionadas como DOM y XSL. También se estudian las APIs de Java relacionadas con esto: JAXP yJDOM.

5. ***"cXML":*** Una vez estudiado con profundidad el lenguaje XML, entramos a estudiar un estándar de comercio electrónico basado en ese lenguaje denominado cXML.

6. ***"JXTA"***: Capítulo dedicado a explicar el API que trata de implementar la tecnología peer-to-peer en el lenguaje de programación Java.

7. ***"Aplicación desarrollada":*** Analiza y explica en profundidad el trabajo desarrollado en este proyecto. Se explica las distintas funcionalidades de la aplicación y se analiza las distintas clases que la componen.

8. ***"Manual de instalación":*** En este apartado se dan las indicaciones necesarias para instalar y ejecutar la aplicación desarrollada. Se dan también unas nociones básicas para configurar JXTA.

9. ***"Planificación temporal del proyecto"*:** Se muestra la planificación del proyecto mediante un análisis de tareas y un diagrama de Gantt.

10. ***"Presupuestos":*** Se estiman los costes de desarrollar este proyecto. Los presupuestos se desglosan en costes de recursos humanos y costes de materiales (costes de software, costes de hardware y costes de consumibles).

11. ***"Líneas futuras":*** En este apartado se apunta posibles direcciones de investigación y aplicación a partir de este proyecto.

12. ***"Conclusiones":*** Se trata de analizar este proyecto en su totalidad sacando las pertinentes conclusiones respecto al resultado obtenido, tecnologías utilizadas…

13. ***"Bibliografía":*** Se citan las principales fuentes bibliográficas, clasificadas temáticamente, que se han consultado a la hora de desarrollar este proyecto.

14. ***"Anexo: Código de la aplicación":*** Se incluye en forma de anexo el código fuente de las clases que componen la aplicación.

Tras esta introducción estamos en condiciones de adentrarnos en la materia que nos concierne, esto es, en el comercio electrónico y en las redes peer-to-peer.

2. COMERCIO ELECTRÓNICO

El comercio electrónico es una materia cada vez más extensa y de difícil enfoque: la literatura sobre el tema está creciendo cada año de forma espectacular, aparecen nuevos estándares y propuestas, existe más información en la red... Abordar este tema en un solo apartado es una tarea muy complicada. Por ello, se ha decidido realizar un pequeño resumen de algunos aspectos básicos del comercio electrónico y de los principales estándares para realizar las transacciones comerciales, ya que es lo que se aborda en este proyecto. Para ampliar información, en la bibliografía aparecen varias obras consultadas que dan visiones distintas sobre el comercio electrónico. En la red es posible encontrar muchísimas páginas con información, aunque de forma más dispersa.

2.1. INTRODUCCIÓN

El comercio electrónico (en inglés *e-commerce*, o también a veces *e-business*) se puede definir, según la Comisión Europea, como "hacer negocios electrónicamente". Esto incluye el intercambio de bienes tanto físicos como intangibles, como la información. Asímismo, comprende todas las etapas de la transacción como el *marketing online*, los pedidos, el pago, soporte para la distribución, servicios post-venta, asesoría legal, etc. También incluye el soporte electrónico para colaboración entre empresas.

El comercio electrónico existe desde hace varias décadas, especialmente con la tecnología EDI (*Electronic Data Interchange*) en sectores como el automovilístico y minorista, y CALS (*Computer Assisted Lifecycle Support*) en la industria pesada y defensa. Sin embargo, estas formas de comercio electrónico han tenido difusión limitada, debido al alto coste de implantación y su dificultad.

A principio de los años 1970, aparecieron las primeras relaciones comerciales que utilizaban una computadora para transmitir datos. Este tipo de intercambio de información trajo consigo mejoras de los procesos de fabricación en el ámbito privado, entre empresas de un mismo sector.

A fines de los años 1970 el Ministerio de Defensa de Estados Unidos inició un programa de investigación destinado a desarrollar técnicas y tecnologías. Así se logró estandarizar las comunicaciones entre computadoras y en 1989 aparece la WWW (World Wide Web, Telaraña

Global). Lo más importante de la WWW es su alto nivel de accesibilidad, que se traduce en los escasos conocimientos de informática que exige de sus usuarios.

El desarrollo de estas tecnologías y de las telecomunicaciones ha hecho que los intercambios de datos crezcan a niveles asombrosos, simplificándose cada vez mas y creando nuevas formas de comercio, y en este marco se desarrolla el Comercio Electrónico.

Se considera "Comercio Electrónico" al conjunto de aquellas transacciones comerciales y financieras realizadas a través del procesamiento y la transmisión de información, incluyendo texto, sonido e imagen.

Dicha información puede ser el objeto principal de la transacción o un elemento conexo a ella. La definición es amplia e incluye cualquier forma de transacción de negocios en las cuales las partes interactúen electrónicamente en lugar de hacerlo en forma física. Establecer contacto con clientes, intercambiar información, vender, realizar soporte pre y posventa, efectuar pagos electrónicos y distribución de productos son algunas de las formas de negociar electrónicamente.

El **comercio electrónico directo** o completo se da cuando todas sus fases se realizan por medios electrónicos, mientras que en el **comercio electrónico indirecto** o incompleto no todas sus fases se completan por medios electrónicos. La confianza es la principal barrera con la que se enfrentan los nuevos medios de pago electrónicos, la gente es reacia a dejar que sus datos bancarios naveguen libremente por la Red. Es importante ganarse la confianza de la gente y para ello es fundamental la información.

El comercio electrónico es un gran nivelador ya que permite que negocios diversos, pequeños o grandes, urbanos o rurales, con mucho o poco presupuesto, acerquen sus posiciones en el mercado; un mercado en formación, en el que el pequeño empresario ha de desempeñar un importante papel en estos momentos o con posterioridad será probable que quede marginado, ya que ese negocio surgirá a costa del suyo propio.

También permite a los consumidores tomar sus decisiones de compra mejor informados al poder consultar catálogos on−line y comparar precios sin salir de casa.

Las transacciones de "comercio electrónico" pueden ser efectuadas, entre los siguientes sujetos:

- **B2B: Entre empresas o " business to business":**

Las empresas pueden intervenir como usuarias −compradoras o vendedoras− o como proveedoras de herramientas o servicios de soporte para el comercio electrónico −v.gr. instituciones financieras, proveedores de servicios de Internet, etc.−.

- **B2C: Entre empresa y consumidor o "business to consumers:**

Las empresas venden sus productos y prestan sus servicios a través de un sitio Web a clientes que los utilizarán para uso particular.

- **C2C: Entre consumidor y consumidor o "consumers to consumers":**

Es posible que los consumidores realicen operaciones entre sí, tal es el caso de los remates en línea.

- **C2A: Entre consumidor y administración o "consumers to administrations":**

Los ciudadanos pueden interactuar con las Administraciones Tributarias a efectos de realizar la presentación de las declaraciones juradas y/o el pago de los tributos, obtener asistencia informativa y otros servicios.

- **B2A: Entre empresa y administración o "business to administrations":**

Las administraciones públicas actúan como agentes reguladores y promotores del comercio electrónico y como usuarias del mismo.

En los últimos años se ha observado un desarrollo inmenso del comercio electrónico. Las dos razones principales son:

- Internet
- World Wide Web
- XML

ya que permiten una gran accesibilidad para todas las empresas, no sólo las grandes, además de la capacidad de difusión que proporciona.

El comercio electrónico sobre Internet está llamado a ser una forma importante de hacer negocios en el futuro. Entre 1996 y 1999 se ha calculado que ha crecido un mil por ciento cada año. El crecimiento más espectacular y prometedor se prevé que suceda en el campo de comercio entre empresas, lo que se ha llamado ***b2b (business-to-business)***. Es quizá en este campo donde el comercio electrónico ofrece un futuro más impactante, ya que permitiría que cualquier empresa accediese a esta forma de comercio, dados sus bajos costes de implantación y sus altos beneficios en poco tiempo. Se prevé que en el presente estén involucradas 245.000 compañías europeas y 630.000 estadounidenses en el comercio electrónico b2b con todo lo que ello supone. Se espera que en breve la mayor parte del comercio electrónico en la red sea b2b, desbancando al comercio ***b2c*** (business-to-consumer) que es el desarrollado con el cliente final.

2.2. CATEGORÍAS DE LAS TRANSACCIONES EN EL COMERCIO ELECTRÓNICO

La Organización para la Cooperación y el Desarrollo Económico ha diferenciado la siguientes veintiséis categorías de transacciones típicas de comercio electrónico.

1. Procesamiento de pedido electrónico de productos tangibles.
2. Pedido electrónico de productos digitalizados.
3. Pedido electrónico y descarga de productos digitales a efectos de la explotación de los derechos de autor.
4. Actualizaciones y agregados.

5. Software de duración limitada y otras licencias sobre información digital.

6. Programa de computación u otro producto digital de uso único.

7. Hosting de una aplicación − Licencia separada.

8. Hosting de una aplicación − Contrato enlatado.

9. Proveedor de servicios de aplicación (Application Service Provider).

10. Tarifas por licencia para la provisión de servicios de aplicación (Application Service provider).

11. Hosting de sitios web.

12. Almacenamiento de datos.

13. Obtención de datos.

14. Entrega de datos exclusivos o de alto valor.

15. Publicidad.

16. Acceso electrónico a asesoramiento profesional.

17. Entrega de información.

18. Acceso a sitios web interactivos con suscripción previa.

19. Portales de compras en línea.

20. Subastas en línea.

21. Programas de derivación de ventas.

22. Operaciones de adquisición de contenido.

23. Transmisión de audio y/o video en tiempo real por la red.

24. Mantenimiento de programa de computación.

25. Atención al cliente a través de redes de computación.

26. Información técnica.

2.3. EL COMERCIO ELECTRÓNICO EN ESPAÑA

2.3.1. Introducción

Para hacer un estudio de la situación actual del comercio electrónico en España se ha recurrido al informe elaborado, con una periodicidad trimestral, por la Comisión del Mercado de las Telecomunicaciones (CMT) sobre el comercio electrónico en España a través de entidades de medios de pago, que abarca el período comprendido entre el primer trimestre de 2001 y el cuarto trimestre de 2004, si bien se centra específicamente en este último. Tiene por objeto cuantificar los volúmenes y la evolución real del comercio electrónico en España a partir de los datos sobre las transacciones electrónicas que se realizan a través de las entidades de medios de pago, más en concreto mediante tarjetas bancarias.

Para ello se utiliza la información suministrada por las entidades de medios de pago españolas, en virtud del convenio firmado entre la Comisión del Mercado de las Telecomunicaciones y las organizaciones de medios de pago: Sermepa-Servired, Sistema 4B y la Confederación Española de Cajas de Ahorro (CECA-Sistema Euro 6000), por el que dichas entidades acordaron en junio de 2000 proveer a la CMT de forma voluntaria, con periodicidad trimestral, los datos relativos a las transacciones económicas en operaciones de comercio electrónico que tengan España como punto de origen y/o destino.

En concreto, se analizan las transacciones comerciales entre sitios web de comercio electrónico españoles que tengan instalado un terminal de punto de venta virtual (TPV virtual) y personas físicas o jurídicas que utilizan como medio de pago tarjetas bancarias emitidas por entidades financieras españolas, y transacciones de no residentes que realizan operaciones en sitios web españoles con un terminal punto

de venta virtual español.

Por tanto, las cifras ofrecidas en ese Informe son un censo de la totalidad de las transacciones comerciales electrónicas, cuyo pago ha sido realizado a través de tarjetas bancarias de pago adscritas a las mencionadas entidades de medios de pago colaboradoras y a través de terminales de punto de venta virtuales.

Aunque otras formas de pago son utilizadas en las transacciones de comercio electrónico, tales como el pago contra reembolso y la transferencia bancaria, el pago a través de tarjetas bancarias es el medio que más se acerca a la esencia del comercio electrónico. Si bien el informe no cubre la totalidad del volumen de comercio electrónico, ya que no incluye los datos relativos a las transacciones comerciales a través de Internet abonadas a través de medios de pago distintos a tarjetas bancarias, sí puede decirse que abarca una parte relevante del comercio electrónico con origen o destino en España.

Por lo tanto, en virtud de las cifras manejadas en ese estudio, puede afirmarse que los datos presentados no constituyen un registro completo de operaciones de comercio electrónico realizadas a nivel minorista, pero sí representan buena parte del total y sirven como muestra representativa a nivel minorista para detectar tendencias y cambios importantes en las distintas clasificaciones.

Las operaciones de comercio electrónico analizadas se desglosan en base a los siguientes criterios: facturación total; número de transacciones; origen/destino geográfico; y rama de actividad económica afectada por la transacción (para esta clasificación utilizamos la CNAE: Clasificación Nacional de Actividades Económicas).

En relación al criterio de origen/ destino de transacción, los datos se agrupan en los siguientes tipos de transacciones comerciales:

- **Transacciones comerciales de España con el exterior**: son operaciones de compra efectuadas con alguno de los medios de pago de clientes de los firmantes del convenio (España), en sitios web de comercio electrónico que utilizan un terminal punto de venta virtual de otro país.

- **Transacciones comerciales del exterior con España:** son operaciones de compras realizadas desde el extranjero utilizando medios de pago en sitios web de comercio

electrónico que emplean terminales de punto de venta virtuales de medios de pago firmantes del convenio (España).

- **Transacciones dentro de España:** son operaciones de compra realizadas con medios de pago de clientes de los firmantes del convenio (España) en sitios web de comercio electrónico que utilizan un terminal punto de venta virtual de medios de pago firmantes del convenio (España).

2.3.2. El comercio electrónico en cifras.

Con un volumen de negocio, de 260 millones de euros – frente a los 243 millones del trimestre anterior – el comercio electrónico en España a través de tarjetas bancarias ha aumentado un 6,9% en el período que transcurre de octubre a diciembre de 2004. Otro tanto cabe decir del número de operaciones efectuadas este trimestre: más de 3,5 millones, con un crecimiento intertrimestral también del 14,6%.

Así pues, el comercio electrónico a través de tarjetas bancarias de pago en España en 2004 continúa su evolución positiva y creciente: con 890 millones de euros ha duplicado los resultados alcanzados en 2003.

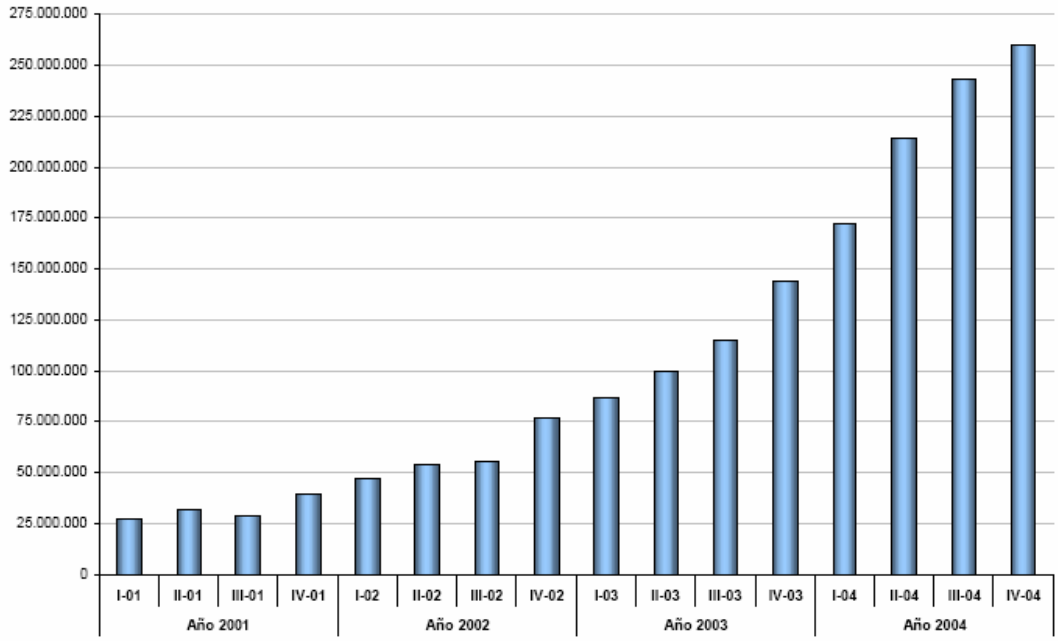


Figura 1: Evolución trimestral del volumen de comercio electrónico en España (Euros)

El montante económico generado durante el cuarto trimestre de 2004 se distribuye principalmente entre los siguientes sectores de actividad: transporte aéreo (26,2%), marketing directo (13,5%) y agencias de viaje y operadores turísticos (5,6%).

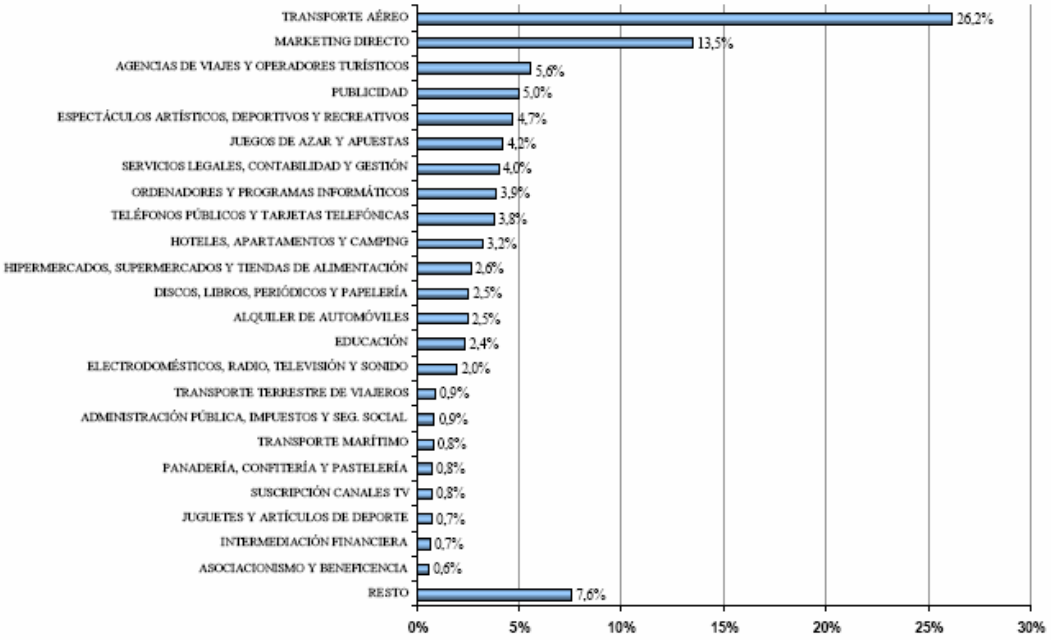


Figura 2: Distribución del volumen de negocio del comercio electrónico en España por sectores de actividad (IV-04) (porcentaje)

De entre los quince primeros sectores por volumen de negocio, ordenadores y programas informáticos es el sector que más ha crecido, triplicando su cifra de negocio entre el cuarto trimestre de 2003 y el cuarto trimestre de 2004, seguido de agencias de viaje y operadores turísticos, transporte aéreo, servicios legales, contabilidad y gestión, y teléfonos públicos y tarjetas telefónicas que duplican sus ingresos.

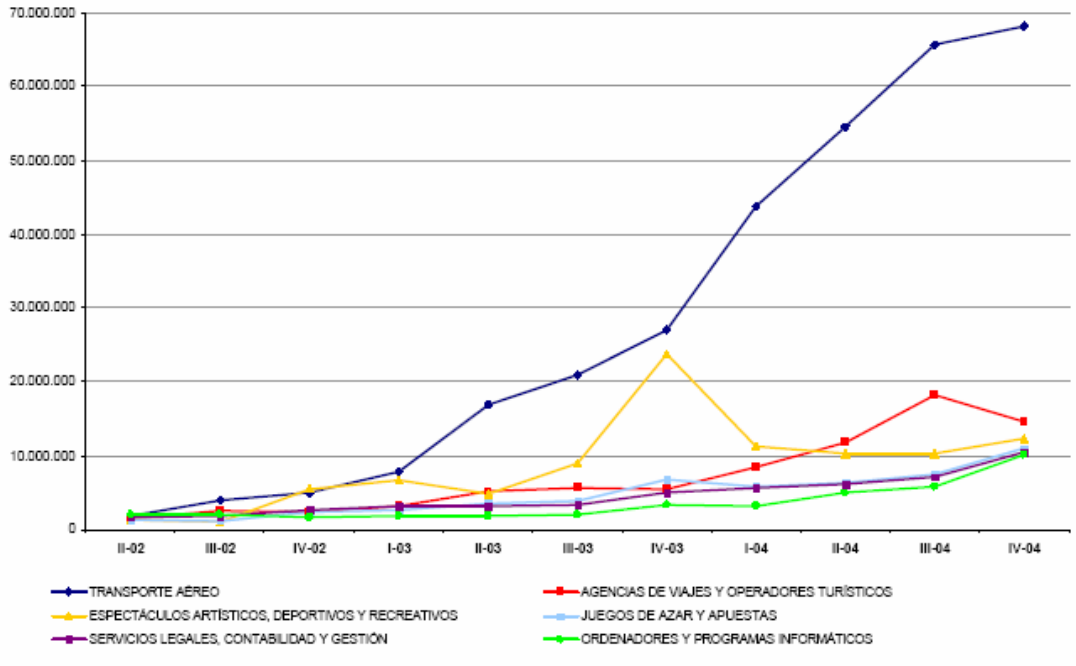


Figura 3: Evolución trimestral de los sectores de actividad más significativos por volumen de negocio del comercio electrónico en España (euros)

El número de operaciones también ha crecido notablemente en 2004, un 78%, alcanzándose a final de año la cifra de 11,6 millones de operaciones. Durante el cuarto de trimestre de 2004 se han realizado un total de 3,5 millones de operaciones de comercio electrónico, un 14,6% más que en el tercer trimestre de 2004. No obstante, si bien la evolución interanual del número de transacciones ha sido positiva, no lo ha aumentado en la misma proporción a la cifra de negocio, sino un 65%.

En función del origen y destino de las transacciones electrónicas, la mayor parte de ellas corresponde a adquisiciones de residentes en España a establecimientos de comercio electrónico situados en el exterior.

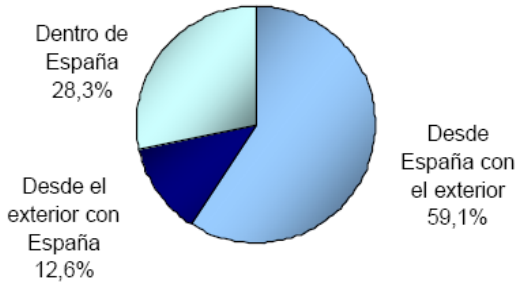


Figura 4: Volumen de negocio del comercio segmentado geográficamente (IV-04)

En 2004 el volumen de negocio de las transacciones con origen en España y dirigidas hacia el exterior ha sido de 520 millones euros – el doble que en 2003 – de los cuáles, al cuarto trimestre del año corresponden 153 millones de euros, que supone el 59% del importe total, un 9% más que el trimestre anterior, con un crecimiento interanual del 88%.

En el cuarto trimestre de 2004, la mayor parte de los importes de dichas compras se ha dirigido a la Unión Europea (80%) con 123 millones de euros y EEUU (13%) con 20 millones.

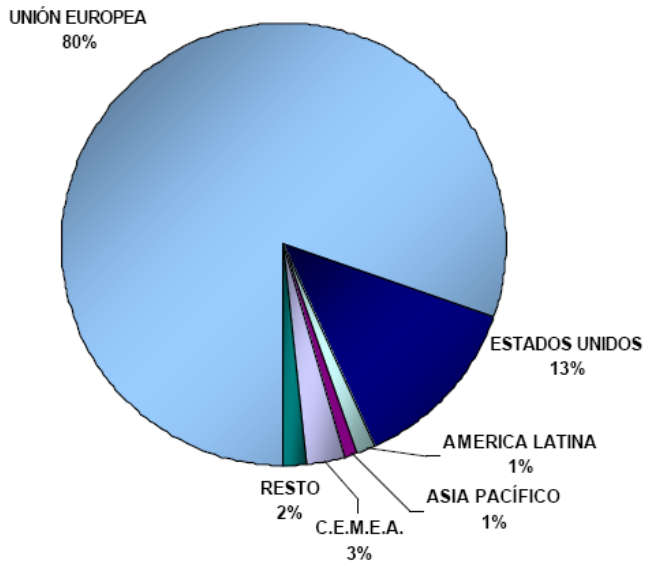


Figura 5: Distribución del volumen de negocio del comercio electrónico desde España con el exterior por áreas geográficas (euros)

El mayor crecimiento trimestral por volumen de negocio se ha experimentado en las compras realizadas desde España dirigidas al área Asia Pacífico –en la que destacan países como Australia, Hong Kong (China), Japón, y Singapur – un 24% más que el trimestre anterior, representando el 1% del total de la cifra de negocio con 1,4 millones de euros.

Hacia los países CEMEA – Países de Europa Central, Oriente Medio y África, entre los que se incluyen los diez nuevos países adheridos a la UE, Suiza, Rusia, Sudáfrica, Turquía y Arabia Saudita – se han realizado compras por valor de 4 millones de euros (3% del total).

Por sectores, son transporte aéreo (27%), marketing directo (11,8%), juegos de azar y apuestas (6,8%), publicidad (6,8%) y servicios legales, contabilidad y gestión (6,7%) los principales destinatarios por

volumen de negocio de las compras españolas en el exterior en el 4º trimestre de 2004.

Respecto a las transacciones realizadas desde el exterior y dirigidas a sitios web españoles, en 2004 su volumen de negocio ha sido de 120 millones de euros, el doble que en 2003, para un total de 1,1 millones operaciones. La Unión Europea, con 160.000 transacciones y un volumen de negocio de 18 millones de euros, sigue siendo el área geográfica que más compra en España a través de Internet (54% del total) en el 4º trimestre de 2004. No obstante, su cifra de negocio ha caído un 21% respecto al trimestre anterior, si bien es un 104% más que en el mismo trimestre de 2003. Esto se explica por la disminución del volumen de negocio desde países como Reino Unido, Alemania, Irlanda o Suiza durante dicho trimestre.

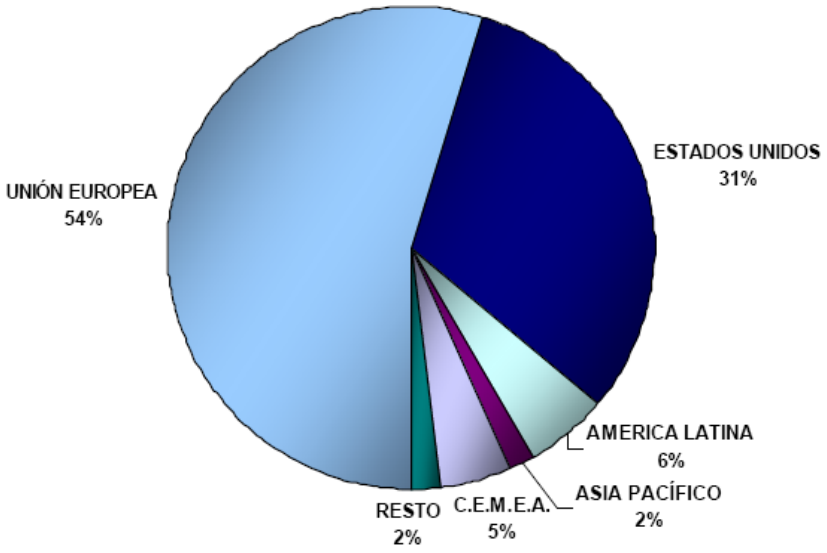


Figura 6: Distribución del volumen de negocio del comercio electrónico desde el exterior con España por áreas geográficas (euros).

El volumen de negocio generado desde EEUU se sitúa en 10,3 millones de euros, un 69% más que el trimestre anterior. Si a este dato añadimos el aumento del 32% en el número de operaciones, podemos intuir una ligera recuperación de las preferencias del público estadounidense con respecto a los productos españoles. El mayor crecimiento interanual del volumen de negocio se ha producido en las compras desde Estados Unidos con un aumento de más de tres veces.

Por sectores, marketing directo ha representado el 28% del total del volumen de negocio en el cuarto trimestre de 2004; le siguen: transporte aéreo (19%), y agencias de viajes y operadores turísticos (10%). Por otro lado, ordenadores y programas informáticos es el sector

que más ha crecido en el último año, multiplicando por doscientos su volumen de negocio respecto del año anterior. También aumentan más que notablemente: marketing directo y transporte aéreo, que quintuplica y cuadriplica respectivamente su cifra de negocio. En relación con el número de operaciones dirigidas a nuestro país con origen en el exterior, el sector ordenadores y programas informáticos es el que ha tenido un mayor incremento: aumentando su resultado más de setecientas veces respecto del mismo trimestre del año anterior.

Con respecto a la cifra de negocio del comercio electrónico generado en España y dirigido a puntos de venta virtuales radicados igualmente en España, en 2004 la cifra de negocio resultante es de 250 millones de euros para un total de 5,1 millones de operaciones.

Entre octubre y diciembre de 2004 este tipo de transacciones (con origen y destino en España) han crecido ligeramente: un 9%. Con relación al número de operaciones, ha habido una variación trimestral positiva si bien exactamente igual a la acaecida en el trimestre procedente: un 12,5%. Por tanto, el volumen de negocio del comercio electrónico con origen y destino dentro de España se cifra en 73 millones euros para un total de 1,6 millones de operaciones en el cuarto trimestre de 2004.

Los sectores de actividad destinatarios de las compras a través de Internet dentro de España en función de los ingresos son: transporte aéreo (26,9%), espectáculos artísticos, deportivos y recreativos (14,8%), marketing directo (10,6%) y teléfonos públicos y tarjetas telefónicas (9,8%).

2.3.3. Consideraciones Generales

Con un volumen de negocio, de 260 millones de euros – frente a los 243 millones del trimestre anterior – el comercio electrónico en España a través de tarjetas bancarias ha aumentado un 6,9% en el período que transcurre de octubre a diciembre de 2004. Otro tanto cabe decir del número de operaciones efectuadas este trimestre: más de 3,5 millones, con un crecimiento intertrimestral también del 14,6%.

Así pues, el comercio electrónico a través de tarjetas bancarias de pago en España en 2004 continúa su evolución positiva y creciente: con 890 millones de euros ha duplicado los resultados alcanzados en 2003.

Por otro lado, el número de operaciones también ha crecido notablemente en 2004, un 78%, alcanzándose a final de año la cifra de 11,6 millones de operaciones. Sólo en el cuarto trimestre se realizaron 3,5 millones de transacciones, un 14,6% más que en el trimestre anterior.

La mayor tasa crecimiento del volumen de negocio que del número de operaciones indica un incremento del valor medio por operación, probablemente relacionado con una adquisición progresiva de una mayor confianza entre los usuarios de los sistemas de seguridad utilizados en el comercio electrónico, que les lleva a realizar operaciones de mayor cuantía.

En la evolución del comercio realizado a través de Internet por períodos trimestrales observamos cómo en este cuarto trimestre de 2004, si bien tanto los ingresos como el número de operaciones históricamente han sido crecientes, respecto del volumen de negocio puede observarse convexidad de su crecimiento en los tres últimos trimestres, esto es, las tasas de crecimientos han sido consecutivamente menores que en el correspondiente trimestre precedente; así, 24,2%, 13,8% y 6,9%, respectivamente.

Por sectores económicos, el peso principal está en el comercio de bienes y servicios que no llevan asociados costes complementarios a la transacción electrónica y en los que la información es más completa o fácil de adquirir; en concreto todos los bienes y servicios relacionados con los viajes (agencias, transporte y alojamiento), compra de entradas para espectáculos, juegos de azar y recarga de tarjetas telefónicas. Así, transporte aéreo, transporte por carretera de viajeros, alquiler de automóviles, agencias de viajes y operadores turísticos, y hoteles, apartamentos y camping, suman aproximadamente casi 50% del volumen de negocio y en general tienen asociadas las mayores tasas de crecimiento interanual: 152% transporte aéreo, 162% agencias de viaje, 80% alojamiento y 65% alquiler de automóviles.

Debido a que el cuarto trimestre abarca los meses de octubre, noviembre y diciembre – esto es, temporada baja y el final del periodo estival – puede apreciarse la caída de la actividad en los sectores relacionados con los viajes debido a la. Por el contrario, se acentúa el crecimiento de sectores como educación (35%, frente al 6% del trimestre precedente) y servicios legales, contabilidad y gestión (45%, frente al 17%)

Igualmente, cabe señalar el fortísimo avance en los últimos trimestres de las compras en materia audiovisual (68%) e informática

(que duplica interanualmente su cifra de negocio).

Este aumento se debe fundamentalmente al incremento de las compras de descargas legales de música a través de Internet y la compra de software y juegos online.

2.4. VENTAJAS DEL COMERCIO ELECTRÓNICO DEL FUTURO

La introducción de Internet en el comercio electrónico ha supuesto una revolución. Las ventajas que ofrece son innumerables, entre las que podemos destacar:

- **Mayor base de clientes potenciales:** los límites geográficos ya no existen, y el crecimiento de usuarios de Internet sigue aumentando espectacularmente cada año.

- **Mayores ingresos:** al tener más clientes potenciales, da la posibilidad de aumentar los ingresos.

- **Ahorro de costes:** la implantación y mantenimiento de sistemas de comercio electrónico es mucho más barata que con otros sistemas como EDI.

- **Salida al mercado muy rápida:** los productos y servicios ofrecidos pueden ser listados en pocos segundos ante millones de usuarios potenciales.

- **Mejora de las relaciones con el otro participante:** permite mantener relaciones interactivas con el cliente u otros participantes, sin importar la localización geográfica ni la hora.

- **Respuesta rápida al cliente:** con un acceso de 24 horas al día, los clientes pueden pedir información y recibirla de forma automatizada en segundos, o en pocos minutos si la respuesta es generada por humanos.

- **Aumenta la transparencia en el mercado:** el desarrollo de agentes inteligentes permite obtener información contrastada con otras empresas similares, fomentando la libre y justa competencia.

- **Autoservicio:** pemite eliminar intermediarios en la cadena de negocios, posibilitando la actuación directa con la empresa proveedora de servicios, con el consiguiente ahorro de tiempo y dinero.

- **Solicitud de información extra de los productos o servicios:** se puede obtener más información mediante técnicas multimedia destinadas a convencer al cliente del producto o servicio ofertado.

Existe otra tecnología relativamente nueva que también está revolucionando la forma de realizar el comercio electrónico: el XML (*eXtensible Markup Language*). Se han desarrollado en pocos años gran cantidad de propuestas basadas en XML para estandarizar las relaciones comerciales de forma electrónica. Entre las muchas ventajas que ofrece XML frente a técnicas tradicionales como EDI se encuentran:

- Interoperabilidad entre sistemas de comercio electrónico heterogéneos.

- Posibilidad de ampliación y adaptación.

- Desarrollo muy sencillo.

- Los documentos XML son legibles tanto para humanos como máquinas.

- Integración perfecta con la web.

El comercio electrónico del futuro, por tanto, estará fundamentado sobre dos grandes pilares:

- Internet

- XML

2.5. ESTÁNDARES DE COMERCIO ELECTRÓNICO

A continuación se presentan las propuestas más importantes que han surgido como estándares para el comercio electrónico:

2.5.1. EDI

EDI (*Electronic Data Interchange*) se remonta a la década de los setenta donde cada empresa usaba su propio formato para el intercambio interno de documentos relacionados con sus procesos productivos. Ante la necesidad de interoperabilidad entre empresas, se crearon dos estándares compatibles en gran parte:

- ANSI X 12 en América, evolucionando hacia redes entre clientes y proveedores que mueven muchos billones de dólares.

- UN/ ECE GTDI (United Nations Economic Commission for Europe Guidelines for Trade Data Interchange) en Europa.

Ambos estándares se unificaron en 1986 bajo el nombre de UN/EDIFACT (Intercambio Electrónico de Documentos para la Administración, Comercio y Transporte) bajo el auspicio de las Naciones Unidas, y aún hoy en día sigue bajo la tutela de este organismo internacional.

Existen varias definiciones interesantes para EDI:

- Comercio sin papel.

- La transferencia de datos estructurados, usando estándares de mensajes acordados, de un sistema de computadoras a otro, por medios electrónicos. (definición dada por IDEA, *International Data Exchange Association*).

- La comunicación inter-empresa y entre computadoras de transacciones de negocios en un formato estándar que permita al receptor interpretar dicha transacción.

Actualmente hay mas de 240 mensajes desarrollados, que cubren una amplia gama de transacciones comerciales, de servicio y administrativas.

EDI aportó en su momento grandes ventajas:

- Se acorta el tiempo de generación de documentos.

- Reducción de costes en todos los niveles.
- Eliminación de errores.
- Rapidez en las respuestas.
- Integración en la cadena de suministros.
- Facturas exactas y precisas.
- Posibilidad de pagos electrónicos.
- Aumento de oportunidades de negocio.
- Bajada de *stock* en almacenes.

Muchas empresas apostaron en principio por EDI, y lograron conseguir altos beneficios; pero otras muchas no pudieron debido al elevado coste que suponía la implantación: se necesita un estudio completo de viabilidad, donde se reduzcan los procesos de la empresa a modelos y se implanten con EDI mediante herramientas de gestión. Entre otros problemas para implantar EDI destacan:

- Formato muy complicado.
- Servidores de precio muy elevado.
- Usa redes de servicios añadidos, por lo que el coste de transmisión de mensajes se dispara.

Muchas empresas han optado por implantar sólo ciertos procesos con EDI, pero pierde efectividad, puesto que EDI es realmente una solución global para la empresa, no parcial.

Como ejemplo, se muestra un fragmento de orden en formato EDI:

```
ISA*00*     *00*  *08*61112500TST      *01*DEMO WU000003
*970911*1039*U00302000009561*0*P?
GS*PO*611125001I*WU000003 *970911*1039*9784*X*003020
ST*850*397822
BEG*00*RE*194743**970911
REF*AH*M109
REF*DP*641
REF*IA*000100685
DTM*010*970918
N1*BY*92*1287
Nl*ST*92*87447
.....
```

EDI supuso un gran adelanto en su momento. Sin embargo, está totalmente desfasado (aunque se sigue usando mucho), su implantación es inabordable para muchas empresas, y no es flexible y adaptable a todos los sectores empresariales. La solución a todos estos problemas la introdujo la tecnología XML, sobre la cual se basan la mayoría de estándares de comercio electrónico.

2.5.2. EDI-XML

Es un propuesta de fusión del EDI tradicional con tecnologías XML, de forma que se implementen los formatos de elementos y mensajes EDI mediante XML, creando un nuevo marco de trabajo de intercambio de datos.

2.5.3. BizTalk

Es una iniciativa de Microsoft basada en XML. Aunque en principio no es un sistema cerrado por completo, pues posee una arquitectura y mensajes abiertos, está completamente ligado al uso de los sistemas operativos y bases de datos de Microsoft. Realmente es más un producto de gestión de procesos de negocio que un estándar. Proporciona herramientas software para el desarrollo de las soluciones, y admite múltiples protocolos, como SMTP, FTP, HTTP, EDI, etc. Su principal objetivo es lograr la integración de aplicaciones y la conectividad entre participantes del negocio. Está disponible en www.biztalk.org.

Los documentos BizTalk son mensajes que siguen el protocolo SOAP, de forma que el documento de negocio es el cuerpo del mensaje.

BizTalk es una gran herramienta, pero es totalmente propietario y limitado al entorno de productos de Microsoft. Las aplicaciones BizTalk deben ejecutarse obligatoriamente sobre productos Microsoft. Además, requieren una gran cantidad de recursos del sistema, y los mensajes son muy complejos, a pesar de la existencia de herramientas visuales para su desarrollo.

2.5.4. RosettaNet

RosettaNet es un consorcio de compañías del sector de las Tecnologías de la Información, Componentes Electrónicos y Fabricación de Semiconductores, cuyo objetivo es crear estándares abiertos para procesos de negocio electrónicos. Entre estas empresas se encuentran IBM, Intel, NEC, Microsoft, y otras. Estos estándares forman un lenguaje común para negocios electrónicos, facilitando así la comunicación entre socios. Está disponible en www.rosettanet.org.

RosettaNet proporciona un marco de desarrollo de soluciones de comercio electrónico, es decir, que no es un producto concreto, sino una infraestructura básica para la construcción de sistemas de comercio electrónico aportando las normativas necesarias para garantizar la compatibilidad entre sus participantes. Sus fundamentos son:

- Diccionario de negocio
- Diccionario técnico
- Marco de trabajo de implementación
- Procesos de interfaz con el participante

Los diccionarios definen cuáles son los términos usados en el proceso, y lo que significan. En RosettaNet existen dos diccionarios: uno define los términos de la transacción en sí y el otro para describir los productos o servicios abarcados en la transacción.

2.5.5. xCBL

xCBL (XML Common Business Library) es un conjunto de módulos XML y modelos de documento, que permite la creación, de forma rápida y reutilizable, de documentos XML para comercio electrónico. El proyecto está liderado por Commerce One, y está apoyado por numerosas empresas como Sun, Microsoft, HP, y otras. Su uso es totalmente gratuito, y las especificaciones están disponibles en www.xcbl.org.

Actualmente está disponible la versión 4.0, que proporciona una migración desde el comercio basado en EDI. XCBL soporta todos los documentos y transacciones para comercio electrónico global, como automatización de cadenas de suministro, subastas, facturas, etc. La versión actual está disponible únicamente en XML Schema, aunque las previas estaban definidas bajo SOX Schemas, un lenguaje propietario de esquemas de Commerce One, lo que impedía su difusión al público general.

Como ejemplo, a continuación se muestra un fragmento de orden de compra xCBL:

```
      <OrderRequest
xmlns:core="rrn:org.xcbl:schemas/xcbl/v4_0/core/core.xsd"
xmlns="rrn:org.xcbl:schemas/xcbl/v4_0/ordermanagement/v1_0/ordermanage
ment.xsd"         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="rrn:org.xcbl:schemas/xcbl/v4_0/ordermanagement/v1_
0/ordermanagement.xsd
../../schema/org/xcbl/path_delim/schemas/xcbl/v4_0/ordermanagement/v1_
0/ordermanagement.xsd">

          <OrderRequestHeader>

              <OrderRequestNumber>

                  <BuyerOrderRequestNumber>REF-002-99-0-
3000</BuyerOrderRequestNumber>

                  <SellerOrderRequestNumber>REF002-
44556677</SellerOrderRequestNumber>

              </OrderRequestNumber>

              <OrderRequestIssueDate>2003-01-
01T00:00:01</OrderRequestIssueDate>

              <OrderRequestReferences>

                  <ContractReferences>

                      <core:Contract>
```

```
                            <core:ContractID>

                                <core:Agency>


<core:AgencyCoded>AssignedByPartyOriginatingTheMessage</core:AgencyCod
ed>

                                </core:Agency>

                                <core:Ident>CONTRACT:002-91</core:Ident>

                            </core:ContractID>

                            <core:TypeOfContract>
<core:ContractTypeCoded>FixedPriceIncentive</core:ContractTypeCoded>

                            </core:TypeOfContract>
```

2.5.6. UBL

Ante la gran cantidad de soluciones XML existentes para llevar a cabo negocios electrónicos, OASIS ha formado un grupo de trabajo con el objetivo de crear un lenguaje XML común para el desarrollo de negocios electrónicos, con la colaboración de los principales organismos de estandarización y empresas líderes en el sector. Este lenguaje se basa inicialmente en xCBL 3.0, y será mejorado teniendo en cuenta la experiencia acumulada en el uso de otras soluciones XML.

2.5.7. ebXML

ebXML (Electronic Business XML) es una iniciativa para la aplicación de XML al comercio electrónico, con el soporte de la ONU y OASIS. ebXML supera las limitaciones comunes del EDI, tales como la falta de un estándar común y la necesidad de sistemas propietarios con características especiales, que se debían replicar en los sistemas de los agentes o socios de la cadena de negocios. Está disponible en www.ebxml.org.

ebXML cataloga los procesos comunes de los negocios y establece una serie de mensajes comunes, que permiten interoperabilidad entre los sistemas de las empresas proporcionando seguridad y consistencia en las operaciones comerciales electrónicas. Las especificaciones proporcionadas incluyen una arquitectura técnica, esquema de especificación para los procesos de negocios, registro de servicios y mensajes. ebXML ha sido diseñado para ser el equivalente actual del EDI, pero de más fácil implantación.

Es necesario recalcar que no se debe confundir ebXML con la tecnología básica que éste utiliza, sino que solamente es una herramienta que ayuda a implementar la lógica comercial.

El fundamento de la arquitectura del ebXML es lograr que los registros y repositorios sean interoperables, mediante un marco de trabajo comercial común que permite que los participantes puedan encontrarse entre sí y sean capaces de generar nuevas relaciones comerciales. El modelo de Registro ebXML está diseñado para ofrecer un repositorio compartido que almacena perfiles comerciales para las compañías participantes. Tras la conexión a un registro compatible con ebXML, las aplicaciones de la compañía serán capaces de acceder a la información de los procesos comerciales, así como también a los modelos de información usados por sus posibles socios comerciales.

El éxito del ebXML radica en que su mecanismo de transporte de mensajes debe tener un modelo de entrega que permita que las aplicaciones envíen los mensajes sin necesidad de manejar admisiones y con un modelo efectivo de seguridad. Al utilizar la especificación del Servicio de Mensajería ebXML, está capacitado para implementar cualquier nivel de protocolo de mensajería, incluyendo SMTP y HTTP, y el protocolo SOAP (utilizado para ofrecer servicios web).

2.5.8. cXML

cXML es un lenguaje estándar, abierto y sencillo, que permite a compradores, proveedores, intermediarios, etc. comunicarse entre sí en un marco de comercio electrónico. Está diseñado, principalmente, para las necesidades de B2B. Su principal impulsor es la empresa Ariba, aglutinando a muchas empresas de todo el mundo. Está disponible en www.cxml.org.

cXML se describe extensamente más adelante en esta memoria. Sólo se mencionará aquí que es un lenguaje muy sencillo e intuitivo, y que la especificación sólo define el lenguaje en sí, a diferencia de los otros estándares comentados anteriormente, que constituyen toda una filosofía de enfoque de negocios, incluyendo procesos de negocio, mecanismos de transporte en ciertos casos, etc. Esto permite que cXML posea, entre otras, estas características:

- Muy fácil de implementar.

- Total libertad a la hora de elegir tecnologías subyacentes.

- Muy sencillo de entender, ya que es parecido a los documentos impresos.

A continuación se muestra un fragmento de una orden de compra usando cXML:

```
      <?xml version="1.0" encoding="UTF-8"?>

      <!DOCTYPE cXML SYSTEM
"http://xml.cXML.org/schemas/cXML/1.2.009/cXML.dtd">

      <cXML payloadID="3223232@ariba.acme.com"

            timestamp="1999-03-12T18:39:09-08:00" xml:lang="en-US">

          <Header>

              .......

      <OrderRequest>

                    <OrderRequestHeader orderID="DO1234"
orderDate="1999-03-12" type="new">

                        <Total>

                            <Money currency="USD">2.68</Money>

                        </Total>

                        <ShipTo>

                            <Address>

                                <Name xml:lang="en">Acme</Name>

                                <PostalAddress name="default">

                                    <DeliverTo>Joe Smith</DeliverTo>

                                    <DeliverTo>Mailstop M-
543</DeliverTo>

                                    <Street>123 Anystreet</Street>

                                    <City>Sunnyvale</City>

                                    <State>CA</State>

                                    <PostalCode>90489</PostalCode>

                                    <Country isoCountryCode="US">United
States</Country>

      .......
```

Con esto se concluye el capítulo dedicado al comercio electrónico en el que se ha pretendido dar una visión general de cuál es su estado actual en España y de cuáles son los principales estándares de transacciones electrónicas.

3. PEER-TO-PEER

La tecnología *peer-to-peer* se ha divulgado ampliamente, y está siendo reconocida como un modelo de red con un futuro muy prometedor. Incluso se está considerando la arquitectura *peer-to-peer* como un modelo de negocio. Es por ello que se dedica este capítulo al estudio de los aspectos generales de este tipo de redes. Al final del apartado se entra al estudio detallado de dos de estas redes: Freenet y Gnutella.

3.1. QUÉ ES PEER-TO-PEER

3.1.1. Definición

En general, una red informática entre pares (conocida en inglés como *peer-to-peer* o P2P) se refiere a una red que no tiene clientes y servidores fijos, sino una serie de nodos que se comportan a la vez como clientes y servidores de los demás nodos de la red, en contraste con el modelo de red cliente-servidor. Cualquier nodo o peer puede iniciar o completar una transacción compatible con la red. Los nodos pueden diferir en configuración local, velocidad de proceso, ancho de banda de la red y capacidad de almacenamiento, dando lugar a una red con integrantes muy heterogéneos.

La idea básica es que dos o más dispositivos comparten entre sí información y recursos, cada uno actuando simultáneamente como cliente y servidor.

3.1.2. Requisitos de un sistema P2P

Otra posible definición de P2P ha sido propuesta por David Winer de Userland Software, en base a los requisitos que debe cumplir un sistema para que pueda ser considerado como P2P:

- Las computadoras usuarias pueden actuar tanto como clientes como servidores.

- El sistema en su conjunto está bien integrado y es fácil de usar.

- El sistema contiene herramientas para ayudar a los usuarios en la creación de nuevos contenidos o nuevas

funcionalidades.

- El sistema proporciona conexión con los otros usuarios.

- El sistema realiza algo nuevo o de utilidad.

- El sistema soporta protocolos interred como SOAP o XML-RPC.

3.1.3. Lo que no es P2P

En P2P existen muchos mitos que distorsionan el verdadero concepto. Entre ellos destacan:

- **P2P no es Napster**: Se tiende a identificar P2P con conocidas aplicaciones de compartición de ficheros basadas en P2P como Napster, GNUtella, Kazaa, Emule, etc. Sin embargo, estas aplicaciones sólo utilizan la arquitectura P2P, pero no son P2P en sí mismas. El P2P es una arquitectura de red completa, sobre la que se construyen aplicaciones.

- **P2P no es compartir archivos**: sobre la arquitectura P2P se pueden construir todo tipo de aplicaciones, no sólo de compartición de archivos: mensajería, almacenamiento compartido, entornos colaborativos, computación distribuida, etc.

- **P2P es un concepto totalmente nuevo**: P2P es un paso más en el avance por la descentralización de las redes. Este concepto es tan antiguo como las redes mismas, y se viene investigando desde hace más de 20 años en universidades y laboratorios, aunque en entornos reducidos y con poca repercusión. La fama actual del P2P se debe a la aparición reciente de aplicaciones de compartición de ficheros, aunque quizá gran parte de esa fama pertenezca más a los escándalos legales que a las aplicaciones en sí.

- **P2P es computación distribuida**: P2P no es ni computación distribuida, ni computación *grid*, ni redes ad-hoc: es un modelo más, aunque se está haciendo esfuerzos por integrar estas tecnologías entre sí .

3.1.4. Características de una red P2P

- Los clientes son servidores y también *routers*, y contribuyen a la red con sus propios recursos como CPU, espacio en disco, etc.

- Los peers son autónomos, sin ninguna autoridad administrativa (aunque puede haberla).

- La red es dinámica: los nodos entran y salen de ella frecuentemente.

- Los peers colaboran directamente entre ellos, sin intermediarios.

- Los peers tienen un amplio rango de capacidades que pueden variar con el tiempo.

3.1.5. Ventajas del P2P

Como ventajas se puede destacar:

- **Escalabilidad:** una red P2P puede contener desde 2 máquinas conectadas hasta millones, pudiendo además agruparse en comunidades interconectadas entre sí.

- **Capacidad de proceso y almacenamiento:** en vez de disponer de un servidor con una limitada capacidad de proceso y almacenamiento, se pueden tener millones de usuarios, cuyas capacidades exceden miles o millones de veces la del servidor sólo.

- **Disponibilidad de servicios y redundancia de datos:** si un peer que presta un servicio se cae de la red, existen otros muchos que también pueden prestarlo, por lo que el servicio sigue disponible. Aumenta, por la misma razón, la redundancia de datos, puesto que puede haber varios peers que poseen dicho dato.

- **Descentralización.**

- **No hay limitación geográfica:** peers situados en zonas geográficas distantes pueden estar próximos en la red P2P.

- **Máximo aprovechamiento de los recursos:** como ancho de banda, CPU, etc.

- **Fácil administración**: debido a que los peers se autoadministran.

3.2. MODELOS P2P

El P2P no es un modelo rígido, sino que admite distintas variantes respecto al modelo puro, gracias a que es una arquitectura abierta y basada en la descentralización. Una posible clasificación de modelos P2P es la siguiente:

- **Atomístico**: es la arquitectura P2P por excelencia, la más ortodoxa, debido a que la conectividad es verdaderamente cliente a cliente, sin presencia de ningún servidor. Sin embargo, por esta razón, no dispone de métodos de creación de enlaces de comunicaciones basados en disponibilidad de datos e identidad de usuario.

- **Centrado en el usuario**: se utiliza un directorio para facilitar las comunicaciones entre los usuarios.

- **Centrado en los datos:** permite a los usuarios buscar y acceder a datos contenidos en sistemas de otros usuarios.

- **Web MK 2**: se trata de la convergencia de los dos últimos modelos citados, con arquitectura e infraestructura Web. Los navegadores evolucionan hacia gestores de espacio de trabajo configurables por el usuario y que integran los citados modelos P2P. Múltiples servidores de directorio se encargan de conectar a los usuarios, y múltiples índices permiten acceder a los datos, tanto si están en servidores como en clientes.

- **Centrado en el proceso:** en vez de usar un único procesador de gran potencia, el proceso de una aplicación se divide entre múltiples clientes, de cuya coordinación se ocupa un servidor.

3.3. P2P VS CLIENTE-SERVIDOR

P2P ofrece una clara alternativa a la arquitectura tradicional cliente-servidor. Aunque P2P emplea las existentes redes, servidores e infraestructuras cliente, su modelo de computación es totalmente ortogonal al modelo cliente-servidor. Sin embargo, esto no quiere decir que sean incompatibles entre sí: ambos modelos deben coexistir y complementarse.

En el modelo cliente-servidor, el cliente realiza peticiones al servidor al que está conectado. El servidor, que está esperando la recepción de dichas peticiones, realiza la operación pedida y puede responder al cliente con otro mensaje de respuesta. El modelo es fuertemente centralizado, pues el éxito de la petición depende de la existencia y disponibilidad del servidor.

En el modelo P2P, cada nodo o peer funciona como cliente, pero sobre una capa de funcionalidad de servidor. Esto permite al peer actuar tanto como un cliente o servidor dentro del contexto de una aplicación. Un peer puede iniciar peticiones y también responder a peticiones de otros peers de la red. Esta posibilidad de la comunicación directa entre los usuarios libera a los peers de la dependencia de servidores centralizados, permitiendo que los peers tengan un elevado grado de autonomía y control sobre los servicios que utilizan en la red.

La Figura 7, Figura 8 y Figura 9 ilustran la diferencia entre modelos cliente-servidor y el P2P. También muestran cómo ambos modelos pueden coexistir en uno híbrido, aprovechando las ventajas ofrecidas por las dos arquitecturas:

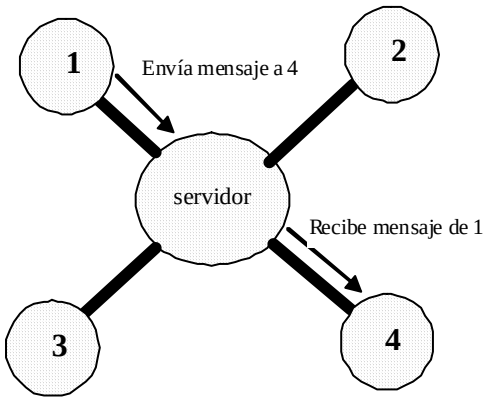


Figura 7: arquitectura cliente-servidor

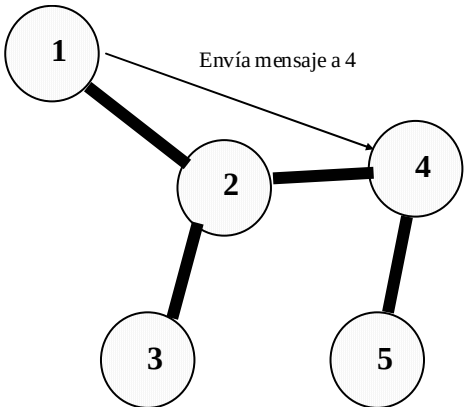


Figura 8: arquitectura P2P

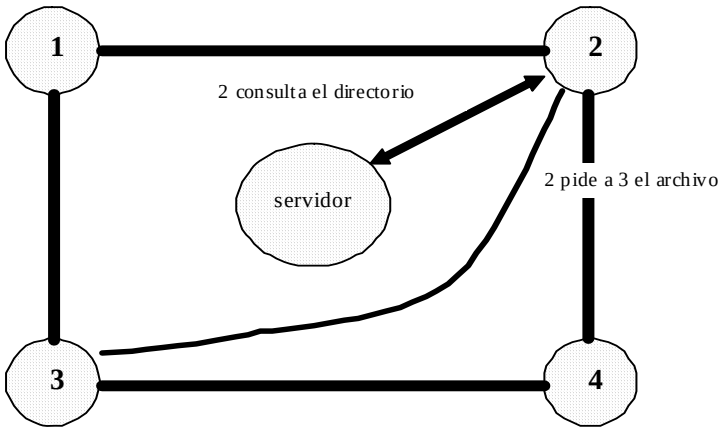


Figura 9: arquitectura híbrida P2P y cliente-servidor

3.4. TIPOS DE REDES P2P

Aunque es difícil hacer una clasificación de los tipos de redes que implementan la arquitectura P2P, se pueden distinguir dos grupos, coincidentes con las dos generaciones que han aparecido:

3.4.1. Sistemas P2P no estructurados

En general coinciden con las redes usadas para el intercambio de ficheros a través de la red. Se basan en el elevado número de usuarios conectados, por lo que la probabilidad de encontrar un determinado

archivo es muy elevada. Se distinguen varios subtipos, atendiendo a la centralización y a su vez de la evolución temporal:

- **Red "Napster"**: los usuarios conectados envían su lista de archivos que compartirán a un servidor centralizado. El usuario que quiera buscar un determinado archivo (de música en este caso), realiza una petición con determinadas palabras clave al servidor, que le devuelve las direcciones IP de usuarios que poseen el fichero buscado. El usuario se conecta a continuación de manera directa con los poseedores del archivo. Este modelo es fuertemente centralizado, puesto que exige la participación de un servidor, sin el cual no se podría realizar ninguna descarga. La presencia del servidor además es la que motiva los problemas legales de la compartición de archivos, puesto que actúa como intermediario de la transacción. Otros problemas son: posible denegación del servicio, cuellos de botella en el servidor, etc.

- **Red "GNUtella"**: la búsqueda, al contrario que en Napster, es descentralizada: se pregunta a los vecinos por el archivo buscado, éstos a su vez a los vecinos , etc, hasta que los poseedores del archivo responden al origen de la petición, iniciándose, al igual que en Napster, la conexión directa entre usuarios. Este mecanismo de búsqueda se basa en inundación, es distribuido pero no muy escalable.

- **Red "FastTrack"**: usada por los clientes Kazaa, Kazaa Lite, iMesh, Grokster. Es una solución híbrida basada en super-peers: ciertos nodos de la red (super-peers) actúan como servidores centralizados, pero sólo en pequeñas porciones de la red. Un usuario registra en el super-peer su lista de archivos, y realiza peticiones de búsqueda al super-peer.

3.4.2. Sistemas P2P estructurados

Se consideran la segunda generación de redes P2P. Sus características más importantes son:

- Auto-organizativas.

- Balancean la carga.

- Tolerantes a errores.

- Número de saltos para atender una petición escalable.

- Basados en interfaz de tabla *hash* distribuida (DHT).

DHT es sólo una interfaz: las implementaciones más conocidas actualmente son: CHORD, PASTRY, TAPESTRY, CAN. Los objetivos de DHT son, en líneas generales, una eficiente inserción/búsqueda/borrado de parejas (clave, valor) y que los nodos almacenen sólo un conjunto de pares sobre otros nodos.

3.5. CAMPOS DE APLICACIÓN DEL P2P

El P2P ofrece nuevos campos de aplicación gracias a su particular arquitectura, pero también ofrece nuevos enfoques para situaciones tradicionalmente pertenecientes a otros modelos. Las posibilidades del P2P se multiplican a cada momento, aunque en la actualidad sólo se emplea en un conjunto reducido de ámbitos. Esto no quiere decir que el P2P sea la panacea: no es adecuado para todos los tipos de aplicación: existen ciertos servicios que implementados con el modelo cliente-servidor ofrecen mejores resultados que con P2P. Entre los campos de aplicación presentes y futuros en los que la tecnología P2P se revela como clave se encuentran:

Comunidades web: P2P permite la creación de grupos para reunir a usuarios con intereses comunes, posibilitando la creación de *intranets* con acceso mediante Web por ejemplo.

e-Business: P2P puede añadir nuevas posibilidades, como la conexión de enlaces en la cadena de suministro, la distribución de información, contenidos y software de manera más eficiente, y el mantenimiento de los elementos de información en sus nodos originales, pudiendo recuperarse con algoritmos de búsqueda o de directorio compartido. Además, puede dar una nueva dimensión a los modelos de negocio B2B.

Juegos: una infraestructura P2P proporciona una buena plataforma para el desarrollo de comunidades de juegos *online* que no están controladas por un servidor central. Los desarrolladores pueden entonces centrarse más en las características propias del juego que en el protocolo de comunicaciones.

Motores de búsqueda: la información puede ser mucho más actualizada, ya que con P2P se busca directamente en el espacio donde reside el elemento pedido.

Protección frente a virus: Las relaciones establecidas entre los nodos de una comunidad P2P permiten la colaboración en la detección y alerta de virus informáticos, así como la prevención automática de toda la comunidad frente a futuros ataques.

Servicios *edge*: En ocasiones es preferible situar los datos más cerca del cliente antes de que éste los pida, reduciendo así el tiempo de espera y la carga de la red. Por ejemplo, una web que ofrezca videos *online* de cocina podría colocar los trozos centrales y de gran tamaño de los archivos en nodos intermedios más cercanos al cliente que el servidor central. Usando P2P, el efecto se puede multiplicar, ya que varios peers que ofrecen espacio de almacenamiento proporcionan un servicio mucho más fiable y flexible que un sólo servidor.

Entornos colaborativos: los campos de aplicación aquí son prácticamente infinitos, entre los más destacados se puede citar:

- Compartición de archivos.
- Desarrollo de software.
- Computación distribuida: como por ejemplo:
    - Simulación.
    - Cálculos numéricos.
    - Control automático.
    - Procesado de señales.

3.6. SEGURIDAD EN P2P

La seguridad es un aspecto esencial en cualquier sistema, y especialmente relevante en los P2P, debido a que su principio de

arquitectura abierta la convierte en un objetivo fácil de ataques, aunque no más que otros sistemas más cerrados y propietarios.

Si no se implementan políticas de protección efectivas, el uso de aplicaciones P2P con entidades externas puede facilitar la entrada de elementos extraños en la red interna, y el sistema seguro se convertirá en uno totalmente vulnerable. Para ello, se deben establecer los medios de actuación en:

- Control de conexión.
- Control de acceso.
- Control de operación.
- Antivirus.
- Protección de datos almacenados.

Los tres primeros son críticos y deben priorizarse sobre los demás, puesto que si se establecen como seguros, los dos restantes no se verían nunca amenazados.

3.6.1. Amenazas externas

Usar redes P2P inseguras permite que la red esté abierta a varias formas de ataques, espionaje, etc, aunque el uso de P2P no implica la aparición de nuevas amenazas, sino la aparición de los ataques tradicionales como virus y gusanos. Afortunadamente, P2P también ofrece mecanismos de defensa muy efectivos frente a estas amenazas, gracias precisamente a su arquitectura abierta: un ataque puede propagarse muy rápido, pero también la cura; pueden existir miles de peers ejecutando políticas de prevención frente a ataques, en vez de ser uno sólo en sistemas más centralizados, en cuyo caso si es atacado el sistema queda desprotegido totalmente; en P2P se pueden formar comunidades seguras aisladas de ataques, impidiendo la propagación.

Algunos de los riesgos externos más significativos son:

- **Ladrones**: programas camuflados que roban información importante.

- **Sobrecarga de la red**: el uso de programas de descarga masiva aumenta mucho los tiempos de respuesta de la red, y puede llegar a colapsarla.

- **Errores**: la descarga de aplicaciones P2P con código defectuoso puede constituir un riesgo de seguridad enorme, puesto que pueden dejar abiertos puertos del sistema, o provocar conflictos entre aplicaciones, e incluso el colapso del sistema, con el consiguiente coste económico que ello supone, especialmente en redes corporativas y de negocios.

- **Desciframiento de encriptaciones**: como P2P permite la computación distribuida a gran escala, es posible usarlo para descifrar claves cifradas con algoritmos de encriptación. En 1999, Distributed.Net lanzó un ataque al algoritmo de encriptación de 56 bits DES, considerado el más potente por el gobierno de los EE.UU.: en menos de 24 horas fue descubierto.

- **Troyanos, virus, saboteadores**: las aplicaciones P2P son instaladas en principio como "de confianza", por lo que se permite el acceso a través de *firewalls*, acceso a los recursos de la máquina (con el consiguiente peligro para datos confidenciales), etc. Cualquier código malicioso podría robar, modificar o eliminar datos, o incluso hacerse con el control de la máquina por completo usando programas puerta trasera (*backdoor)*, y ni siquiera se tendría constancia de ello, puesto que se "confía" en la aplicación.

- **Autentificación, autorización**: en redes P2P, de naturaleza en principio abierta, se deben establecer políticas de autentificación para entrar en la red y autorización para la ejecución de operaciones de riesgo a fin de evitar que usuarios impostores o de no confianza puedan acceder y ejecutar dichas operaciones.

3.6.2. Amenazas internas

Las amenazas internas a las redes P2P son igualmente peligrosas que las externas. Entre ellas se pueden destacar:

- **Interoperabilidad**: es el mayor asunto de seguridad en las redes P2P. La introducción de nuevas plataformas, diferentes sistemas y aplicaciones trabajando juntos dentro de una infraestructura común presenta un conjunto de problemas de seguridad de interoperabilidad. Cuantas más diferencias haya, más complejos serán los problemas.

- **Negocios privados en redes públicas**: muchas organizaciones llevan a cabo negocios sobre redes públicas, lo que expone a muchos riesgos de seguridad.

- **Añadir y eliminar usuarios**: el sistema P2P debe poseer un mecanismo flexible y eficiente para la adición y eliminación de usuarios de la red, pero sin comprometer la seguridad, de forma que se impida, por ejemplo, la adición de usuarios conflictivos.

- **Usuarios maliciosos**: siempre habrá usuarios que intenten acceder clandestinamente a las redes privadas, y tarde o temprano lo consiguen. Se deben mejorar continuamente los protocolos y las políticas preventivas para evitar o retrasar lo máximo posible el acceso, o al menos minimizar daños.

3.6.3. El futuro de la seguridad en P2P

El principal asunto en la seguridad en P2P es la confianza: confianza en los otros usuarios con los que se interactúa, y confianza en el *software* proporcionado. Incrementar la confianza en estos dos aspectos incrementaría a su vez el uso de tecnologías basadas en P2P.

Se han propuesto varias ideas respecto a esto:

- **Los usuarios deben ganarse la confianza de los demás**: es una idea reciente propuesta y llevada ya a cabo en ciertas aplicaciones P2P de compartición: a todos los usuarios (no máquinas) se le asigna un cierto certificado único e intransferible. Este certificado tiene asociado un cierto nivel de confianza: dependiendo del comportamiento del usuario, el nivel crecerá (su confianza ante los demás es alta) o bajará (no es un usuario de confianza). Lo importante es que los certificados se asignan a las personas usuarias, no a las máquinas. Esto obliga a las personas a

comportarse de cierta forma frente a los demás, igual que en la sociedad real; un usuario de poca confianza será excluido por el resto de la comunidad "honesta", y no se le permitirá acceder a los servicios. Sin embargo, esta idea tiene ciertos defectos a la hora de llevarla a la práctica: la falta de una autoridad central permite que los usuarios se conviertan en su propia autoridad, pudiendo alterar los certificados (la solución es usar el modelo híbrido, con un servidor centralizado de certificados de confianza).

- **Biométricas**: consiste en la autentificación en la red mediante el uso de las características personales como huellas digitales, retina de los ojos, etc.

- **Nuevos métodos de criptografía**: el desarrollo, por ejemplo, de nuevos algoritmos criptográficos usando cuántica ofrecería más seguridad a la hora de proteger mensajes y claves de acceso a la red P2P.

3.7. RETOS Y FUTURO DEL P2P

Los principales problemas que presenta la arquitectura P2P se relacionan con aspectos aún no maduros de la tecnología. El avance del P2P dependerá del avance en estos aspectos, entre los que destacan los siguientes:

- Diseño y organización de los sistemas y mecanismos distribuidos de índice que sean eficientes para la obtención de datos.

- Mecanismos de localización y rutado distribuidos que permitan acceso uniforme y disponible a los recursos.

- Manejo de los aspectos dinámicos del sistema, como la admisión y salida de usuarios (nodos) y la publicación y eliminación de los recursos correspondientes.

- Mecanismos de gestión distribuida que agreguen recursos a múltiples peers y satisfagan grados de QoS extremo a extremo y requerimientos de tiempos de respuesta en tiempo real.

- Proveer mecanismos de seguridad y privacidad eficientes, como autentificación, encriptación, etc.

El potencial del P2P ha sido ya probado, y el interés demostrado por la industria es muy elevado, pero aún quedan muchos problemas por resolver, especialmente el de la interoperabilidad entre sistemas. Las aplicaciones P2P no pueden comunicarse de manera fácil con otras aplicaciones P2P, ya que cada una tiene su propio conjunto de servicios y *plug-ins.* Un problema aún mayor es que los desarrolladores de nuevas aplicaciones no pueden beneficiarse de la experiencia de otros desarrolladores P2P. Así, se emplea mucho tiempo en el desarrollo de servicios que ya han sido creados por otros, en vez de haber añadido nuevas funcionalidades a los ya existentes.

Este problema se resolverá cuando la industria u organizaciones mundiales colaboren en el desarrollo de estándares de interoperabilidad ente aplicaciones P2P. Se puede interpretar este conjunto básico de servicios interoperables como una capa *software* intermedia, denominada *middleware.* Con esta capa, las aplicaciones que usen distintos lenguajes de programación y ejecutados en distintas plataformas pueden comunicarse sin ningún obstáculo, y los desarrolladores podrán dedicarse a crear nuevos servicios, o mejorar los ya existentes, en vez de preocuparse por el P2P. Una solución propuesta para el *middleware* es JXTA. En la siguiente figura se representa el modelo de capas de una aplicación P2P:

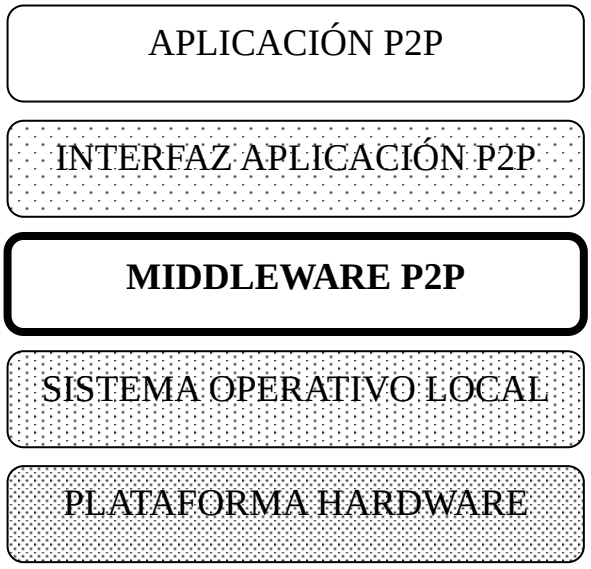


Figura 10: Modelo de capas de aplicación P2P

Se augura un gran futuro para el P2P, aunque queden muchos asuntos pendientes por resolver. Es importante no considerar el P2P como la solución a todos los problemas: es sólo una tecnología más que aporta una visión no excluyente de las redes distribuidas, y su verdadero potencial se manifestará cuando las tecnologías existentes interaccionen y converjan en una más eficiente.

3.8. REDES P2P

3.8.1. Red Freenet.

3.8.1.1. Introduccion.

La red Freenet es un tipo de red montada sobre redes TCP/IP que intenta solucionar varios problemas inherentes a Internet, principalmente la falta de anonimato. Es una red que surge como consecuencia práctica del estudio "A Distributed Decentralised Information Storage and Retrieval System", realizado por Ian Clarke en la Universidad de Edimburgo.

3.8.1.2. El problema del anonimato.

En Internet, el anonimato es una utopía. En redes TCP/IP con el modelo cliente-servidor todos los datagramas IP que llegan a un nodo contienen tanto la dirección IP de origen como la dirección IP de destino.

Anonimato del cliente.

Anonimato del atacante.

Un problema inherente a cualquier ataque a seguridad por parte del atacante consiste en esconder el origen del mismo. Para ello usualmente se utilizan técnicas como el Bouncing o Spoofing, ampliamente conocidas, pero que no solucionan totalmente el problema del anonimato.

El **spoofing** es especialmente útil para ataques a torres de protocolos que no requieren conexión, del estilo de "nukes", o para el "sesión hijacking" y variantes, pero tiene que darse un escenario concreto para poder usarlo.

El **bouncing** permite hacer más difícil encontrar el origen real de

una conexión, pero es posible (con tiempo, medios y un poco de suerte) trazar todos los saltos de una conexión hacia atrás, encontrando al usuario final que lanzo la misma.

Anonimato de un usuario legitimo.

Cualquier usuario que se conecte a un servidor TCP o UDP debe saber que su consulta puede quedar registrada en una base de datos, con todas las posibilidades que ello conlleva de tráfico de información, cosa que es difícilmente detectable y no digamos ya evitable.

Incluso utilizando servidores DHCP que proporcionen una IP distinta cada vez (como ocurre con ciertos ISPs), el servidor DHCP puede controlar dichos datos y almacenar un registro de los mismos, de forma que un ataque a un ISP (cosa que no es tan descabellada, en vista del nivel de seguridad en los mismos, sobre todo debido a sus usuarios) puede proporcionar información sobre los accesos de cualquier cliente del mismo.

Si se utiliza un proxy o NAT, el nivel de detalle que puede obtenerse trazando las conexiones salientes del mismo es menor, ya que no se sabe en principio cual de todos sus usuarios es el que causa el inicio de conexión por parte del proxy, pero se puede tener una idea de las preferencias del grupo, lo cual tampoco está nada mal...

Anonimato del servidor.

Si el anonimato del cliente es difícil, el del servidor es nulo. Todo servidor debe ser accesible a partir de su dirección IP bien conocida, ya sea porque es una IP fija o mediante una traducción DNS en servidores DynDNS (tipo no-ip).

El caso es que toda información publicada en Internet tiene la marca del servidor en que se alberga. Así pues, censurar contenidos en la red es tan sencillo como cerrar el servidor, bien por vía legal, bien por vía intimidatoria.

Problemas de seguridad.

Puesto que es imposible que un servidor sea anónimo a los clientes que a él acceden, atacar al mismo es siempre posible. No hay ningún modo de "esconder" al servidor de forma que los atacantes no puedan encontrarlo. Por ello la seguridad al cien por cien nunca es garantizable.

Esto ocurre con todo tipo de servidores, incluso con servidores DNS o con pasarelas, ambos necesarios para el correcto funcionamiento de la red.

Si se consigue "tirar" un gateway, el segmento de subred al que éste estaáconectado queda inaccesible, y el resto de la red queda a su vez inaccesible para esta subred. Por lo tanto, se producen "cuellos de botella" en cuanto a disponibilidad de la red, que podrían ser subsanados si dichos servidores/pasarelas fueran anónimos.

Soluciones a estos problemas.

Las principales soluciones a los problemas de seguridad consisten en evitar que un ataque sea fructuoso. Esto se hace mediante cortafuegos, parches, políticas de seguridad, concienciación del usuario... También puede albergarse la información en distintos "mirrors", de forma que sea difícil cerrarlos o practicar ataques contra todos, pero otra vez dados tiempo y medios suficientes esto puede no ser eficaz.

Las soluciones a la libre difusión de contenidos por la red se basan hoy en día en conexiones punto a punto entre clientes. En estos sistemas, el servidor no contiene la información, sino son los clientes los que la almacenan y comparten. De esta forma, no es posible cerrar el servidor por almacenar información ilegal ya que este solo se ocupa de conectar a los clientes entre si, en hacer que sean "visibles" unos y otros. Pero aún así, sigue sin haber anonimato, ya que debemos indicar a cada host al que queremos conectarnos cuál es nuestra IP.

3.8.1.3. Funcionamiento de Freenet.

En este apartado intentaremos ofrecer una somera imagen del funcionamiento de la red freenet, sin entrar en muchas honduras. Para un estudio más profundo habría que remitirse a la web oficial, http://freenet.sourceforge.net/, donde se podrá encontrar documentación exhaustiva sobre el tema, amen de software de acceso a la red.

Freenet es anónima tanto para cliente como para servidor.

La red freenet se basa en esta red TCP/IP no anónima y crea una red anónima mediante un sencillo mecanismo. Cada host no se conecta directamente al servidor que contiene la información, sino que se crea una malla de servidores/clientes totalmente descentralizada.

La idea es que cada host actúa tanto de cliente como de servidor y que todos tienen el mismo poder dentro de la red. Cuando un usuario quiere conseguir cierta información, su host comprueba si la tiene. Si no es así, redirecciona la petición a una lista de hosts que tiene. Estos comprueban si la información esta almacenada en ellos, y si no la tienen almacenada a su vez redireccionan la petición a otros hosts.

Una vez se llegue a un nodo que contenga dicha información, este nodo la envía al nodo anterior que la pidió, quien a su vez la manda al anterior que la pidió, y así hasta que se llega al nodo que inicialmente lanzo la petición, el cual pasa la información al usuario.

Lo bueno que tiene este sistema, es que cada nodo recibe peticiones otros nodos "próximos", pero no sabe si el sumidero de información será el nodo que le hizo la petición o si éste está redireccionando una petición de un nodo anterior.

Asimismo, cuando un nodo contesta un mensaje de petición y envía la información, el siguiente nodo que la recibe no sabe si los datos están almacenados en el nodo que se la manda o éste está redireccionando los datos desde otro nodo anterior.

Un ejemplo ayudará a aclarar el mecanismo de funcionamiento. Supongamos que hay un nodo A que, debido a una petición de un usuario, quiere encontrar el archivo "datos.dat". Este nodo está conectado a los nodos B, C y D. El nodo B está a su vez conectado al nodo E, el cual contiene precisamente el archivo "datos.dat", y los nodos C y D están conectados a otros nodos que no tienen dicho archivo.

Cuando el nodo A recibe la petición de su usuario, comprueba si tiene el fichero en su disco duro. Como no lo encuentra, lanza un mensaje de petición a los nodos B, C y D. Los nodos C y D comprueban que no tienen el mensaje e intentan reenviar la petición a otros nodos. Supondremos que el mensaje, que tiene un "tiempo de vida" (al igual que los datagramas IP) muere antes de llegar al nodo E por estos caminos. En cambio, el nodo B reenvía el mensaje al nodo E, el cual si tiene la información pedida. Entonces el nodo E contesta enviando el fichero al nodo B, el cual a su vez lo envía al nodo A. Una vez llega al nodo A, el protocolo de freenet pasa los datos al usuario, y ahí termina el proceso.

Vemos que el nodo B solo sabe que el nodo A quiere "datos.dat"

pero no sabe si quiere el archivo para él o para otro nodo que se lo ha pedido. Lo mismo ocurre con el nodo E: solo ve que B le pide el archivo, pero no sabe si lo quiere para él (de hecho, en este caso lo quiere para A, pero ni el nodo B ni el E saben quien es el receptor final de la información).

Con la respuesta ocurre exactamente lo mismo. B no sabe si E tiene el fichero o si lo está recibiendo de otro nodo y lo está reenviando. A solo ve que B contesta con el archivo pedido, pero no sabe que el nodo que realmente lo tiene es E. De esta forma, el anonimato esta garantizado.

Freenet consigue que la información más valiosa sea más accesible.

En efecto, la información más valiosa en freenet es la más accesible. Lo primero que cabe preguntarse es ¿cuál es la información más valiosa? La respuesta es simple: la que más veces se pide.

Supongamos que cierto archivo es muy solicitado por distintos nodos que conforman la red. Para evitar tener que realizar las costosas (a nivel de recursos de ancho de banda) peticiones que deben ser reencaminadas, cada nodo tiene una "caché". En dicha caché se almacenan los archivos que han sido obtenidos de la red, pero se diferencia de las caches de, por ejemplo, los navegadores en que esta caché sirve a todos los nodos de la red. Esto quiere decir que si un nodo recibe una petición de un archivo que está en su caché, dicho nodo contestará enviando el archivo al que lo pidió. De este modo, puesto que el nodo final que pide el archivo replica el mismo en su caché, la información esta siendo duplicada en cada acceso. Así, si por ejemplo existe cierto archivo muy solicitado, este archivo estará replicado en distintos nodos a lo largo de la red, de forma que para acceder a el será necesario dar menos saltos que si solo estuviera en un nodo.

Además, así se hace virtualmente imposible un ataque de negación de servicio sobre dicha información: ahora esta replicada en distintos nodos desconocidos para el atacante.

Un ejemplo que el autor Ian Clarke explica, es el caso en que un archivo situado inicialmente en EE.UU. sea muy solicitado por usuarios de Japón. Este archivo debe cruzar el cable trasatlántico, lo que supone un fuerte consumo debido a la escasez de ancho de banda del mismo. Pero una vez esta replicado en un nodo en Japón, los nodos japoneses realizaran la petición y serán atendidos por el nodo que la ha replicado

en Japón, en lugar de por el originario estadounidense.

Freenet hace que la información menos valiosa desaparezca.

Hay un pequeño problema con la memoria caché: los discos duros son finitos y no pueden almacenar todos los ficheros que se reciban.

Llega un momento en que habrá que borrar algo de la caché para poder guardar nuevos datos en ella. Y lo que se borra es, precisamente, lo que ha sido menos pedido en un tiempo determinado.

De esta forma, si un archivo deja de ser pedido por los usuarios, éste irá desapareciendo paulatinamente de cada nodo de la red que lo almacene, salvo que vuelvan a realizarse peticiones del mismo.

Un efecto que puede darse es que un archivo se mueva de su ubicación inicial hacia otros nodos más o menos lejanos que tengan alrededor muchos nodos que pidan ese archivo. Por ejemplo, en el caso del nodo japonés anterior, puede ocurrir que dicho archivo sea un texto en japonés. Lo mas normal será entonces que reciba muchas mas peticiones en Japón que en EE.UU., lo cual hará que, con el paso del tiempo, el nodo original estadounidense lo borre de su caché al no recibir suficientes peticiones. En cambio, si el archivo es bastante popular, el nodo japonés y sus colindantes lo consideraran información valiosa, y no la borraran. De esta forma, la información ha "viajado" desde su origen en EE.UU. hasta su nueva ubicación en Japón. Se supone que así se consigue atender a la mayoría de personas, o a minorías lo suficientemente numerosas y localizadas.

El problema son las minorías no localizadas. Si alguien escribe un artículo sobre la cría del galápago en cautividad pero los pocos usuarios interesados se encuentran desperdigados entre Madagascar, Nueva Zelanda, Nepal y Honduras, dicho artículo tiene alta probabilidad de desaparecer de la red.

Los problemas de Freenet.

Todo este sistema teórico parece bastante interesante, pero la cosa no es tan sencilla como parece.

<u>El malgasto de recursos.</u>

El principal problema que tiene Freenet es el despilfarro de recursos de red del que hace gala. Si en una conexión "peer to peer" IP en la que los paquetes dan N saltos de un nodo a otro se consume N x R donde R son los recursos en ancho de banda y tiempo de proceso necesarios para transmitir los datos, en Freenet no es así.

Primero, los datos no van directamente de un extremo al otro, sino que pasan por varios nodos intermedios. Si el número de nodos intermedios es I, se consume entonces teóricamente (en una red en la que los nodos I estén unidos por enlaces directos) I x R, y además I debe ser mayor o igual que N (generalmente, será mucho mayor que N).

Pero es que además la red Freenet está montada sobre una red TCP/IP ya existente, con lo cual entre cada par de nodos intermedios la conexión no será directa, sino que los datos darán un número de saltos determinado por la estructura actual (recordar que el encaminamiento en IP es dinámico) de la red TCP/IP. Si suponemos que el número de saltos entre cada par de nodos es M, los recursos consumidos son entonces I x R x M. Hemos aumentado la cantidad de recursos necesarios en un orden de magnitud.

Una solución a esto, consiste en el ya dicho uso de la caché, de forma que este despilfarro de recursos solo debe darse en la primera petición del archivo. También ocurre que si los datos están en la caché de algún nodo cercano, el número de saltos que hay que dar para llegar a los mismos se reduce, reduciéndose a su vez el impacto de la petición en el resto de la red. Pero esto depende de cuánto se pida el archivo, donde, a que otros nodos este conectado nuestro host, etc. y nunca es una solución mejor que una conexión extremo a extremo.

La inundación.

Otro malgasto de recursos se da debido a que, puesto que no conocemos el host que contiene los datos que queremos, debemos inundar la red de peticiones de forma que lleguen al host adecuado.

Esta inundación es controlada por dos mecanismos: el tiempo de vida y la detección de bucles.

El tiempo de vida de un mensaje, al igual que en IP, es un contador que cada nodo va decrementando al reencaminar dicho mensaje. Una vez llega a cero, el mensaje se desecha y se envía una indicación al nodo que inicio la petición. El problema ahora es que el

nodo que la inició no es conocido, de forma que la única manera de llegar a él es seguir el camino inverso, enviando la indicación al nodo anterior, que la enviará a su anterior, etc. Entonces estamos gastando muchos recursos también en mensajes de control, no solo en los mensajes de datos. Pero esto es siempre mejor que dejar que un mensaje inunde la red y nunca muera, consumiendo recursos de forma ilimitada y, finalmente, saturando la red. También hay que enviar indicaciones de que el mensaje ha muerto al nodo inicial porque si no, nunca sabrá que paso con su petición (y posiblemente la respuesta del mismo sea repetirla, con lo que se inicia de nuevo el proceso de consumo de recursos). Además, el tiempo de vida lo decide el nodo inicial, de forma que si este valor es muy alto, la propagación de la petición puede llegar a ser brutal (recordar que en una red en árbol el crecimiento del número de mensajes reencaminados es exponencial).

La detección de bucles ocurre cuando un nodo recibe una misma petición desde dos nodos anteriores distintos. Esto ocurre porque hay un bucle en la red y este nodo debe tirar uno de estos mensajes (y, por supuesto, no reenviar la petición a ninguno de estos otros nodos). En el modelo de Freenet, la detección de un bucle genera a su vez un mensaje de indicación al último nodo que realizo la petición repetida, de forma que este sepa que la información no va a encontrarse por ese camino (puesto que, si se encuentra, se reenviará al nodo que envió primero la petición duplicada). Nuevo consumo de recursos.

La identificación de datos.

Puesto que los datos pueden estar duplicados, nada impide a un usuario malicioso contestar una petición con datos que no sean realmente los requeridos, sino con otros datos cualesquiera. Para evitar esto hay que utilizar mecanismos de firma digital de los datos. Pero al final las claves públicas para verificar las firmas deben estar almacenadas en algún servidor "confiable", esto es, que sepamos quien nos esta dando estas claves. Por lo tanto la red Freenet no es autónoma y dependerá de otras redes (por ejemplo la World Wide Web) en las que los servidores no sean anónimos.

Las actualizaciones.

Si queremos publicar una versión actualizada de un documento, nos enfrentamos a un problema: existirán versiones antiguas del mismo circulando por la red. ¿Como resolverlo? Mediante la firma digital podremos certificar que somos el creador del documento inicial y que queremos actualizarlo, enviando mensajes de actualización por la red

para que todos los nodos que contengan versiones antiguas las reemplacen por la nueva. El problema es que es imposible asegurar que estos mensajes llegaran a todos y cada uno de los nodos que contengan la versión antigua del documento, y puede ser que haya nodos que sigan manteniendo esta versión aún tras la actualización.

Si estos nodos reciben muchas peticiones del documento, esta versión antigua puede ser replicada muchas veces aunque haya una versión nueva. Incluso podría ocurrir que, debido a que la versión antigua sea más fácilmente accesible que la moderna, esta última desapareciese "asfixiada" por la antigua. Para evitar esto, las nuevas versiones deben inundar la red de mensajes de actualización, volviendo de nuevo al problema del malgasto de recursos.

El software.

El último problema se refiere a la implementación real de la red Freenet, siendo los anteriores de tipo teórico y aplicables a cualquier red de este tipo.

El problema de la Freenet real es que está implementada en Java, lo que supone sacrificar velocidad en favor de portabilidad para una red que ya de por si es lenta.

Además, existen problemas de compatibilidad del cliente/servidor Freenet con distintas máquinas virtuales Java.

Freenet en Internet.

Freenet es un proyecto software albergado en sourceforge (sourceforge.net), es totalmente freeware, la web oficial en sourceforge es:

http://freenet.sourceforge.net/

El cliente/servidor se arranca en nuestro host y permite conexiones directamente, reencaminándolas hacia Freenet. De esta forma, es posible hacer una petición desde el mismo navegador que usamos para la web, puesto que el cliente/servidor actúa como una especie de proxy entre Freenet y nuestro navegador. Además es configurable para que sirva de acceso a otros host (de forma que actúe como "portal" hacia freenet), pero esta forma de uso no está recomendada por los

creadores, puesto que rompe con la filosofía de "totalmente descentralizado" del proyecto, creando posibles vulnerabilidades frente a ataques contra estas "pasarelas".

Además, hay varios programas que permiten otras funcionalidades aparte del acceso Web, como por ejemplo un chat totalmente anónimo.

3.8.2. Red GNUtella.

3.8.2.1. Introducción

Los servicios ofrecidos en Internet, en su mayoría, están implementados bajo el modelo de procesamiento denominado cliente-servidor. A diferencia de este esquema altamente centralizado, Gnutella define una red a nivel aplicación, bajo la modalidad compañero a compañero (peer to peer), donde todo nodo realiza al mismo tiempo funciones de cliente y servidor. Conceptualmente, es un sistema distribuido para almacenamiento y búsqueda de información.

Para comprender las características y el estado de desarrollo del protocolo, resulta necesario remitirse a sus orígenes. Gnutella surgió como un proyecto independiente de los programadores Justin Frankel y Tom Pepper, fundadores de la empresa Nullsoft, destacada por ser la dueña del producto Winamp, reproductor de archivos de sonido mp3. El día 14 de marzo del año 2000, en el sitio Slashdot (http://www.slashdot.com) se publicó una versión, en fase beta, del programa Gnutella; descrito como "Una herramienta de software para compartir archivos, que puede ser aún más potente que Napster". Inmediatamente la página fue puesta fuera de servicio, pero ya habían sido descargadas numerosas copias. Nullsoft, ante el hecho en cuestión, se reservó el derecho de publicación (acción que no volvió a ser realizada) aduciendo que la alta demanda colapsó el sitio.

Actualmente, existe una continuación al proyecto llevada a cabo por una comunidad de programadores. En base a técnicas de ingeniería inversa y análisis de protocolos han logrado decodificar y realizar varias implementaciones (clones) del protocolo Gnutella. El sitio principal de distribución es http://gnutella.wego.com.

Es importante destacar dos características fundamentales que hacen a Gnutella merecedor de estudios por parte de grupos de investigación y empresas: su carácter descentralizado y su espíritu cooperativo. A partir de esto, han surgido diversos proyectos basados

en el modelo compañero a compañero en redes amplias.

Actualmente se está estudiando la posibilidad de optimizaciones a la mecánica de ruteo (propagación de mensajes), recuperación y especificación de consultas. La división Distributed Search Services de la empresa Clip2 (http://dss.clip2.com) llevó a cabo estudios sobre la evolución y condición actual de la red Gnutella y el área Internet Ecologies de la Empresa Xerox realizó un análisis del tráfico en la red, caracterizando el comportamiento de los usuarios.

3.8.2.2. Sistemas Distribuidos Emergentes para Compartir Archivos.

Un Sistema Distribuido para Compartir Archivos (SDCA) permite que usuarios remotos que operan en máquinas distribuidas puedan intercambiar archivos. Si en este modelo cada máquina puede ofrecer una porción de su sistema de archivos ó solicitar por un archivo remoto (cada máquina es igual en funcionalidad), el sistema es considerado como compañero a compañero. Esto permite el desarrollo de nuevas aplicaciones y servicios basados en usuarios finales.

El avance de las investigaciones en esta línea permitirá un mejor aprovechamiento de los recursos informáticos. Por ejemplo, una empresa que posea un importante número de computadores puede combinarlos para potenciar su disponibilidad de recursos (típicamente espacio de almacenamiento y tiempo de CPU) y llevar a cabo una determinada tarea.

Ejemplos de modelos de SDCA son los Freenet, Napster, Scour Exchange…

3.8.2.3. Características de Gnutella

Bajo Gnutella se trabaja en un modelo de ambiente distribuido. Un nodo X ofrece abiertamente los archivos que desee compartir con otros usuarios, mientras que a la vez el usuario dueño del nodo Y puede estar recuperando, como cliente, archivos de un nodo X. Normalmente las tareas de servidor y cliente están siempre activas en un nodo Gnutella, pero nada impide que alguna pueda no ser activada.

La red Gnutella se compone de numerosos nodos distribuidos a lo largo del mundo. Su topología no indica jerarquía alguna, dado que cada nodo es igual a los demás en funcionalidad. Debido a que uno de

los atributos que caracterizan a un nodo es su ancho de banda ofrecido para descarga de archivos, el modelo sufre ciertas variaciones. Aquellos nodos de mayor ancho de banda son preferidos a otros, formando hubs ó anillos centrales de conexión a la red.

Cada nodo de Gnutella sólo sabe acerca de los nodos con los que se conecta directamente Todos los otros nodos son invisibles, a menos que ellos se anuncien contestando a un mensaje tipo ping o contestando a una consulta. Esto proporciona un alto grado de anonimato.

De forma resumida, la red Gnutella opera bajo el modelo conocido como "propagación viral". Un cliente envía un mensaje a un nodo, y éste lo reenvía a todos los nodos a los cuales esté conectado. De esta forma, con solo conocer la dirección de red de un nodo ya se puede ingresar a la misma.

El ingreso a la red se realiza indicando la dirección IP y puerto TCP de cualquier nodo que ya esté conectado. El nodo que inicia la conexión anuncia su presencia mediante el envío de un mensaje. El nodo al cual se conectó reenvía este mensaje a todos los nodos a los cuales esté conectado directamente y así sucesivamente. Cada uno de estos nodos, responde a este mensaje indicando cuántos archivos comparte y tamaño total de los mismos.

Un ejemplo del alto grado de estabilidad e independencia que presenta el modelo puede verse en el siguiente ejemplo: Suponga que un nodo-1 se conecta un nodo-2, todo lo ofrezca nodo-1 puede ser tomado por nodo-2 y viceversa. Ahora nodo-3 se conecta a nodo-2 y éste le informa a qué nodos está directamente conectado (los mensajes de nodo-3 pueden llegar a nodo-1 dado que los reenviará nodo-2). A continuación nodo-2 se pone en estado fuera de servicio. La red sigue funcionando normalmente dado que nodo-3 posee direcciones alternativas de nodos, pasadas anteriormente por nodo-2, que le permitirán conectarse a nodos alternativos para seguir vinculado a la red.

En las implementaciones del protocolo Gnutella un parámetro a definir es la cantidad de conexiones simultáneas a la red que siempre un nodo debe tratar de mantener. Cuantas más conexiones tenga, se logrará un mayor espacio de consulta y se asegurará una alta disponibilidad de la red.

3.8.2.4. Estructura de Mensajes

Un mensaje Gnutella se transporta sobre protocolo TCP. Su cabecera siempre consta de 23 bytes y consiste de los siguientes campos: identificador de nodo (16 bytes), función (1 byte), TTL ó tiempo de vida restante (1 byte), Hops ó saltos (1 byte) y largo de la carga en bytes (4 bytes). El valor referenciado en el campo función indica que tipo de acciones deben realizar aquellos nodos que reciben el mensaje. La cabecera descripta es fija para cualquier tipo de función instanciada, lo que es variable en tamaño es la segunda parte ó carga del mensaje dado que depende directamente de la función. Gnutella para operar necesita cinco funciones que le permitirán mantenerse en la red, consultar por recursos y descargar archivos.

1. Función Ping ó Init (código 0x00)

Se utiliza a los efectos de que nodos anuncien su presencia en la red. No lleva carga de mensaje, es decir que el campo largo de la carga se instancia en cero. Cada nodo que reciba un mensaje Ping debe responder generando mensajes Pong (pueden ser n) donde anuncien su presencia ó de nodos que tengan noticia. Un nodo generará un Ping, además, para recabar información sobre nodos vecinos.

Ejemplo. mensaje Ping

27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62

00 07 00 00 00 00 00

Mensaje

Identificador de nodo : 27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62

Función : 00

TTL : 07

Hops : 00

Largo de la carga : 00 00 00 00

2. Función Pong ó Init_Response (código 0x01)

Un nodo envía un mensaje Pong como respuesta a un mensaje Ping. La carga de un Pong se compone de los siguientes campos: puerto (2 bytes), dirección IP (4 bytes), cantidad de archivos compartidos (4 bytes) y kbytes compartidos (4 bytes). Es interesante destacar que un nodo al recibir datos como los descriptos puede abrir nuevas conexiones que le permitan extender su dominio de acción y, a la vez, darle una mayor estabilidad a su presencia en la red Gnutella. Un nodo puede enviar más de un mensaje Pong como respuesta a un único Ping, dado que informa acerca de otros nodos que tenga almacenado en su memoria caché.

Ejemplo. mensaje Pong

27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62

01 07 00 0E 00 00 00 CA 18 AA D2 62 95 03 00 00

00 00 00 00 00

Mensaje

Identificador de nodo : 27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62

Función : 01

TTL : 07

Hops : 00

Largo de la carga : 0E 00 00 00

Carga

Puerto : CA 18

Dirección IP : AA D2 62 85

Cantidad de archivos

compartidos : 03 00 00 00

Kbytes compartidos : 00 00 00 00

3. Función Push (código 0x40)

Es utilizada por nodos que acceden a la red atravesando un firewall, y no pueden descargar un recurso situado en un nodo al otro lado del mismo. El nodo que requiere el recurso enviará un mensaje Push al nodo que lo tiene, con los siguientes campos: identificador de nodo (16 bytes), índice (4 bytes), dirección IP (4 bytes) y puerto (2 bytes). Una vez obtenido el mensaje, el nodo poseedor del recurso establecerá una conexión con el nodo peticionante y le enviará, a la dirección IP y puerto informado, el archivo referenciado por el campo índice. Una respuesta a la función Push no está definida, dado que un nodo puede enviar el recurso ó no hacer caso de la petición.

Ejemplo. mensaje Push

27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62

40 07 00 1A 00 00 00 22 11 3F 71 49 B2 D4 11 88

23 00 80 AD 40 22 11 02 00 00 00 AA 12 62 AA CA

22

Mensaje

Identificador de nodo : 27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62

Función : 40

TTL : 07

Hops : 00

Largo de la carga : 1A 00 00 00

Carga

Identificador de nodo : 22 11 3F 71 49 B2 D4 11 88 23 00 80 AD 40 22 11

Índice : 02 00 00 00

Dirección IP : AA 12 62 AA

Puerto : CA 22

4. Función Query (código 0x80)

Se utiliza a los efectos de que un nodo pueda consultar a otros nodos por un recurso. La carga utilizada por esta función se compone de dos campos, a saber: velocidad mínima requerida para descarga (2 bytes) y un criterio de búsqueda de longitud variable. Un nodo al recibir un mensaje Query y constatar que no hay coincidencias en su estructura de archivos no generará ningún mensaje de respuesta.

Ejemplo. mensaje Query

28 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62

80 07 00 06 00 00 00 00 00 2A 2E 61 00

Mensaje

Identificador de nodo : 28 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62

Función : 80

TTL : 07

Hops : 00

Largo de la carga : 06 00 00 00

Carga

Velocidad mínima : 00 00

Criterio de búsqueda : 2A 2E 61 00 (*.a + carácter nulo)

5. Función Query_Hit (código 0x81)

Se utiliza como respuesta a un mensaje Query. Los campos que componen un Query_Hit son: cantidad de hits ó coincidencias (1 byte), puerto (2 bytes) y dirección IP de descarga (4 bytes), velocidad de operación (4 bytes), resultados (longitud variable) e indentificador del nodo que responde (16 bytes). Los resultados (que pueden ser n) se estructuran como una lista, de la siguiente forma: índice ó número de respuesta (4 bytes), tamaño del archivo (en kbytes) (4 bytes), nombre del archivo (longitud variable) y delimitador de registro (0x0000). El nodo que responde incluye en la respuesta su identificador a los efectos de que lo pueda utilizar el nodo peticionante, si decidiera solicitar un recurso mediante un mensaje Push.

Ejemplo. mensaje Query_Hit

28 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62

81 07 00 57 00 00 00 03 CA 18 AA D2 62 95 1C 00

00 00 00 00 00 00 70 00 00 00 31 34 39 61 72 63

68 69 2E 61 00 00 01 00 00 00 70 00 00 00 31 34

39 61 72 63 68 69 2E 62 00 00 02 00 00 00 70 00

00 00 31 34 39 61 72 63 68 69 2E 6D 00 00 A0 3D

8D 3D 7D 84 D1 11 85 8E 52 54 00 DC 37 66

Mensaje

Identificador de nodo : 28 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62

Función : 81

TTL : 07

Hops : 00

Largo de la carga : 57 00 00 00

Carga

Cantidad de hits : 03

Puerto : CA 18

Dirección IP : AA D2 62 95

Velocidad : 1C 00 00 00

Estructura resultados

Identificador de nodo : 3D 8D 3D 7D 84 D1 11 85 8E 52 54 00 DC 37 66

Estructura resultados

1er elemento

Índice : 00 00 00 00

Tamaño de archivo : 70 00 00 00

Nombre de archivo : 31 34 39 61 72 63 68 69 2E 61 (149archi.a)

Terminador : 00 00

2do elemento

Índice : 01 00 00 00

Tamaño de archivo : 70 00 00 00

Nombre de archivo : 31 34 39 61 72 63 68 69 2E 62 (149archi.b)

Terminador : 00 00

3er elemento

Índice : 02 00 00 00

Tamaño de archivo : 70 00 00 00

Nombre de archivo : 31 34 39 61 72 63 68 69 2E 6D (149archi.m)

Terminador : 00 00

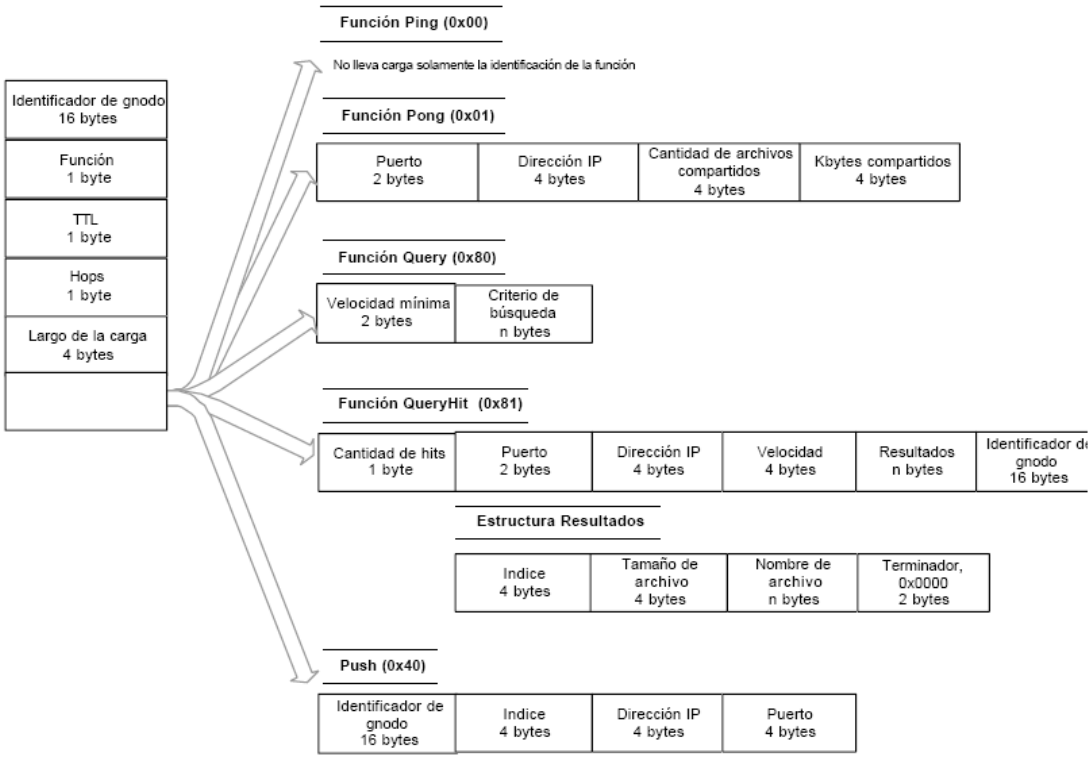


Figura 11: Estructura de los mensajes GNUtella.

3.8.2.5. Reglas de Propagación

El modelo compañero a compañero implantado en Gnutella requiere que los nodos tengan la capacidad de propagar los mensajes recibidos (funciones Ping, Pong, Query, Query_Hit y Push) a través de sus conexiones establecidas. A los efectos de hacer eficiente la operación de la red (teniendo en cuenta su topología, la no jerarquía y la existencia de loops) deben implementarse en cada nodo, una serie de reglas de propagación que a continuación se detallan:

Regla a. Un nodo propagará mensajes tipo Ping y Query a todos sus nodos conectados directamente, excepto al que entregó dicho mensaje.

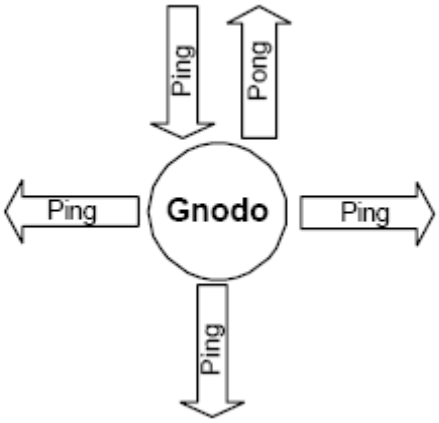


Figura 12: Propagación de Ping

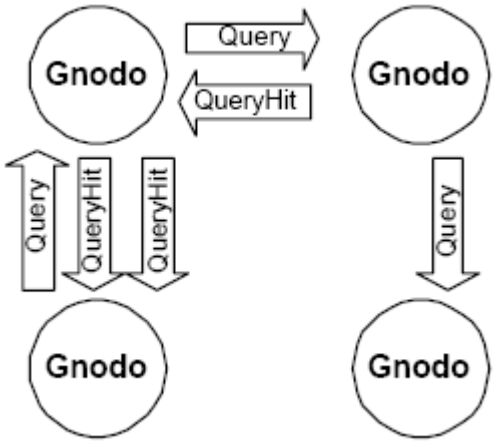


Figura 13: Propagación de consultas

Regla b. Los mensajes tipo Pong, Query_Hit y Push solo deben ser propagados por el mismo camino por el que viajó el mensaje inicial (Ping ó Query) asociado. Esto es posible debido a que antes de propagar un mensaje, los nodos revisan en tablas internas por aquel que generó tal respuesta, y así obtienen la conexión por donde deben reenviarlos.

Regla c. Un nodo decrementará en 1 el valor del campo TTL de un mensaje e incrementará el valor del campo Hops en 1 antes de propagar dicho mensaje por las conexiones pertinentes. Si al decrementar el valor de TTL el resultado obtenido es cero, debe desechar el mensaje.

Regla d. Si un nodo recibe un mensaje con idéntico identificador de nodo y mismo valor de función que uno recibido anteriormente (en un período breve de tiempo, no especificado formalmente) debería evitar propagarlo.

En un instante de tiempo puede haber miles nodos en la red, pero para un nodo en particular su dominio de acción estará restringido a la cantidad de conexiones establecidas y al alcance de sus mensajes (antes de que el valor del campo TTL llegue a cero y sea descartado). Debido al dinamismo, la topología y la existencia de usuarios temporales, es posible que las respuestas a una misma consulta varíen en el tiempo. Cada vez que un usuario ingresa en la red puede hacerlo a una parte diferente de la misma, según el estado actual de sus nodos vecinos.

3.8.2.6. Descarga de Archivos

Una vez que un nodo recibe un resultado (Query_Hit) a una consulta previamente realizada, puede optar por seleccionar un archivo para su descarga desde el nodo remoto. Los archivos se recuperan directamente, sin intervención de nodos intermedios y por lo tanto tampoco de la red Gnutella. La descarga se realiza mediante el uso del protocolo HTTP versión 1.0

Un nodo al recibir un Query_Hit obtiene los siguientes datos: dirección IP y puerto donde el nodo que posee el recurso está en modo de apertura pasiva, índice, tamaño y nombre del archivo.

A los efectos de iniciar la descarga, el nodo que requiere el recurso abrirá una conexión TCP contra el nodo remoto en el puerto anteriormente obtenido y, mediante la primitiva GET del protocolo HTTP, solicitará el recurso seleccionado, indicando índice y nombre de archivo.

Ejemplo

GET /<índice>/<nombre de archivo>/ HTTP/1.0

Connection: Keep-Alive

Range: bytes=0-

El nodo remoto al recibir tal petición y validar que el recurso solicitado está disponible, transmitirá el archivo, previo envió de una cabecera estándar de HTTP con el siguiente formato:

HTTP 200 OK

Server: Gnutella

Content-type: application/binary

Content-length: <tamaño del archivo en bytes>

Es importante destacar que es posible la inclusión de la funcionalidad resumen en la cabecera de la primitiva de protocolo GET, a los efectos de solicitar el reenvío de un archivo a partir de una posición determinada y hasta el final. Esto es útil en caso de descargas previas que hayan sido interrumpidas.

3.8.2.7. Funcionamiento de una Red Gnutella

A continuación se describe el comportamiento de una red Gnutella formada por cuatro equipos que actúan como nodos y sobre los que corre el software Gnutella versión 0.56 (http://gnutella.wego.com). A continuación se describe cada uno:

Nodo 2.

Dirección IP del nodo: 170.210.98.2 (en gráfico se hace referencia al último byte)

Identificador de nodo: 4E62 (cuatro últimos bytes)

Archivos compartidos: 2archi.a, 2archi.b y 2archi.z

Nodo 3.

Dirección IP del nodo: 170.210.98.3

Identificador de nodo: XXXX

Archivos compartidos: 3archi.a, 3archi.b y 3archi.i

Nodo 150.

Dirección IP del nodo: 170.210.98.150

Identificador de nodo: 4D60

Archivos compartidos: 150archi.a, 150archi.x y 150archi.p

Nodo 149.

Dirección IP del nodo: 170.210.98.149

Identificador de nodo: 3760

Archivos compartidos: 149archi.a, 149archi.b y 149archi.m

En los gráficos, cada flecha indica el sentido del mensaje, y la identificación de la misma consta de cuatro elementos: # de mensaje, tipo de mensaje, identificación del origen y TTL. La flecha gruesa indica una conexión abierta y su sentido (desde el punto de vista de la apertura). Los tipos de mensaje están codificados de la siguiente manera

Cn: GNUTELLA CONNECT– Apertura de conexión a nivel aplicación

Ok: GNUTELLA OK – Aceptación de apertura de conexión

Pi: PING – Ping

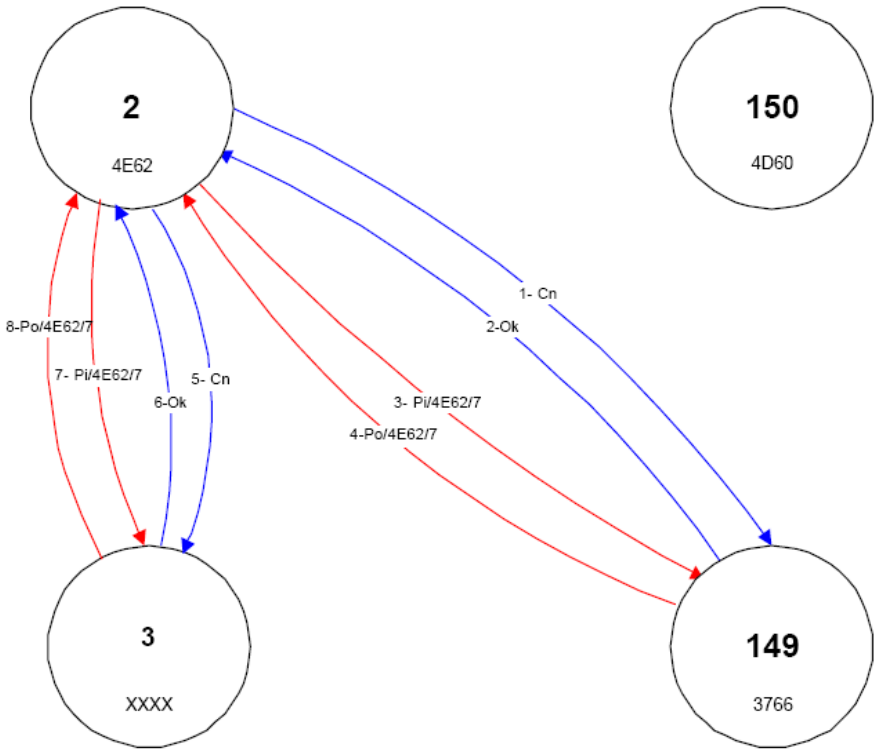
Po: PONG – Pong

Qy: QUERY – Consulta

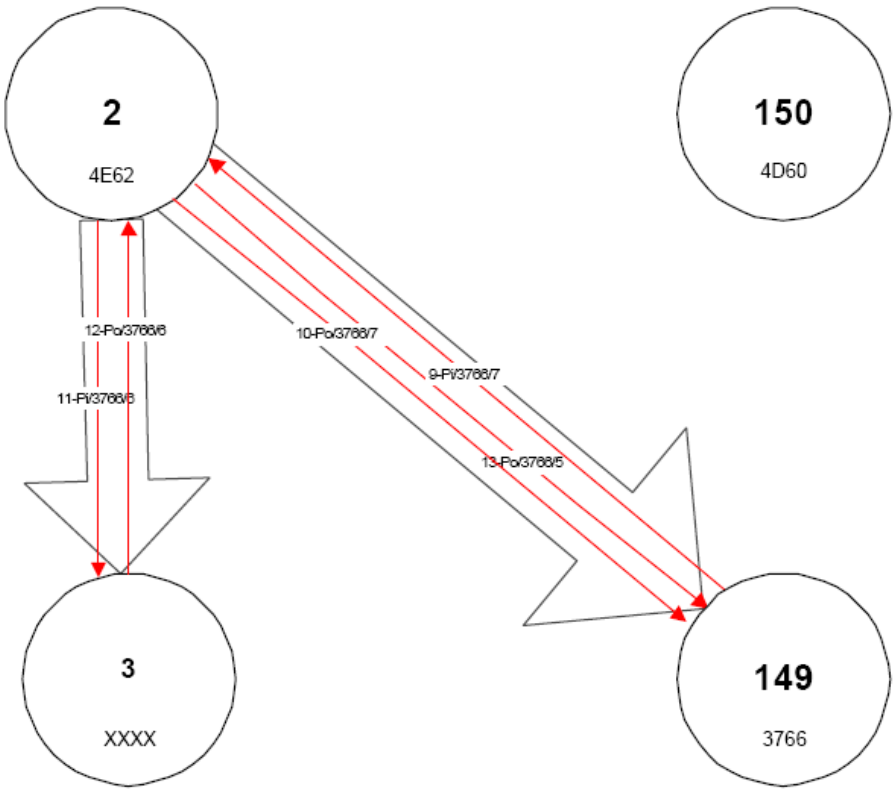
Rs: QUERY_HIT – Respuesta

La secuencia de operaciones fue la siguiente:

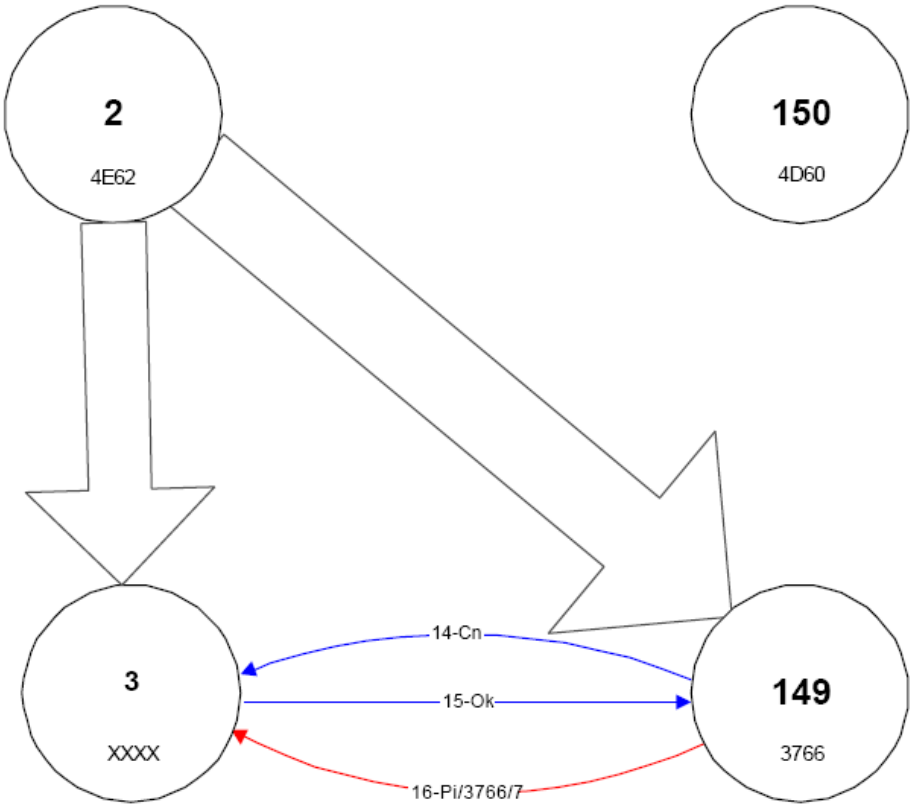
El primer lugar, se solicitó al nodo 2 la apertura de comunicación con el nodo 149 y luego con el nodo 3. Una vez aceptada la apertura de conexión a nivel aplicación, el nodo 2 envió un mensaje Ping a cada uno de ellos, obteniendo un mensaje Pong como respuesta.



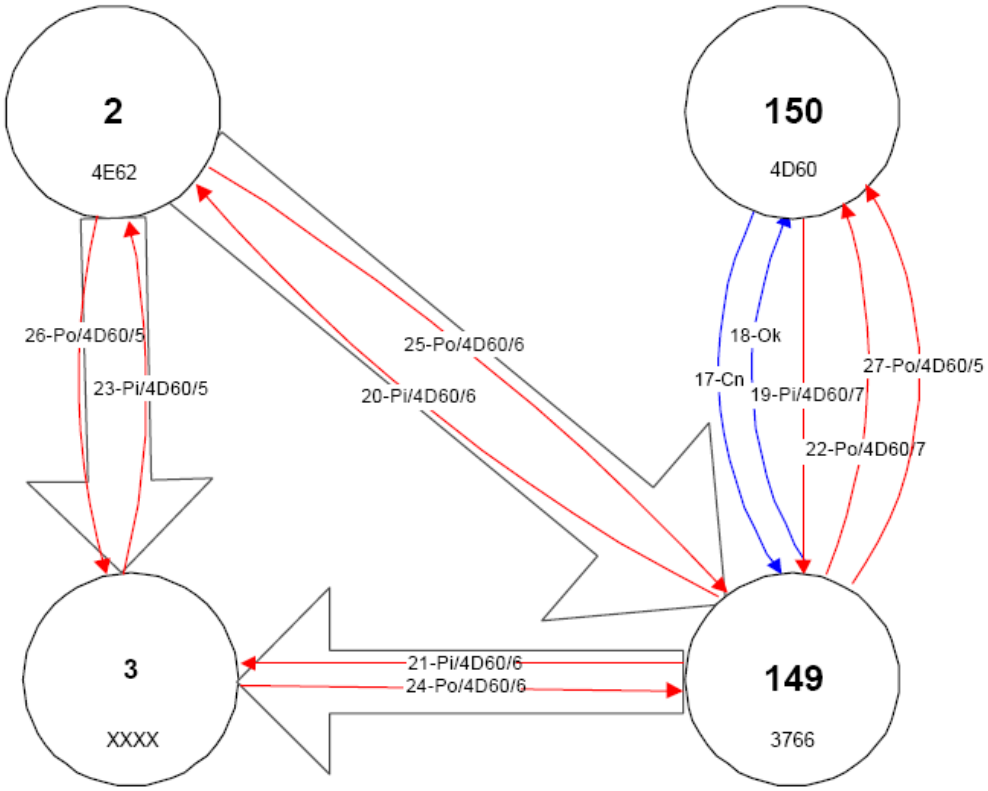
A continuación, se solicitó al nodo 149 la ejecución de un comando de aplicación de actualización de la información de conexiones Gnutella y recursos compartidos, lo que generó el envío de un mensaje Ping por todas sus conexiones abiertas. Como solo tenía conexión con el nodo 2, le envió el Ping a éste, quien primero lo contestó (con su mensaje Pong) y luego lo propagó por todas sus conexiones abiertas, excepto por donde lo recibió (en este caso con el nodo 3), decrementando el TTL del mensaje. Nodo 3 contestó al nodo 2 al Ping y éste lo redireccionó al nodo 149, que fue quien originó el requerimiento.



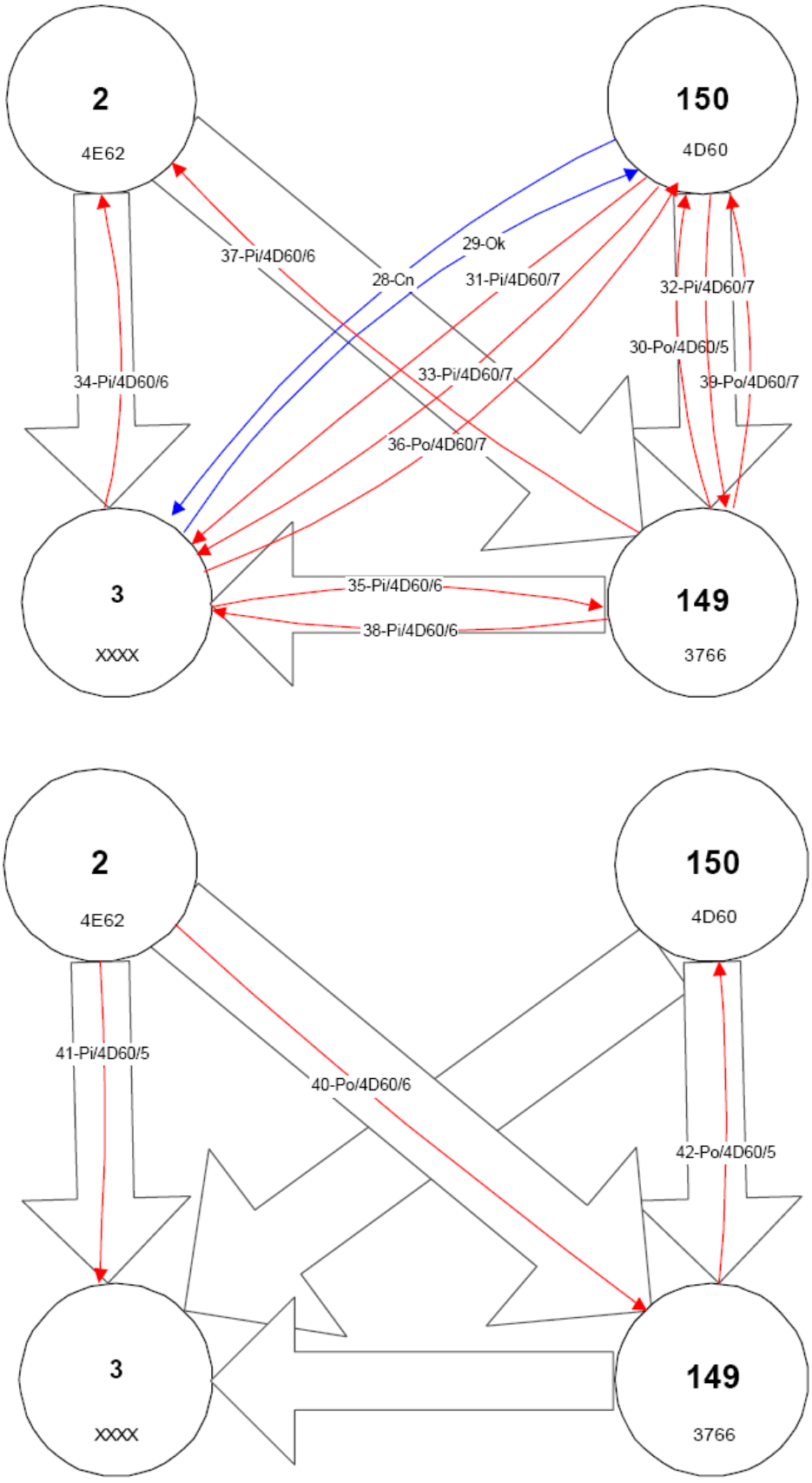
Al recibir información de la existencia del nodo 3, el nodo 149 abrió una conexión con éste y le envió un mensaje Ping .



Luego, se solicitó al nodo 150 la apertura de una conexión con el nodo 149. Finalizada la misma, el nodo 150 envió un mensaje Ping al nodo 149, el cual lo contestó (Pong) y luego lo propagó por todas sus conexiones abiertas (con los nodos 2 y 3, quienes a su vez hicieron lo propio). Las respuestas viajaron por los mismos caminos por donde circuló el mensaje Ping y, finalmente, llegaron al nodo 150 (con el TTL del mensaje decrementado según los nodos que se atravesaron en el camino).

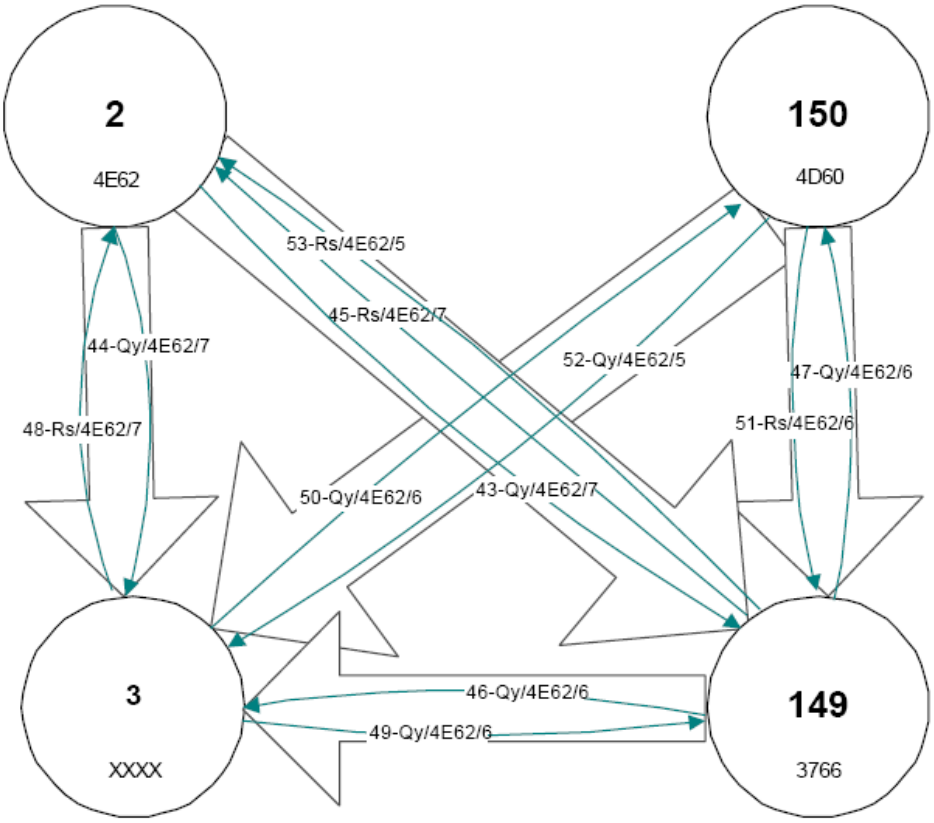


Al recibir información de la existencia del nodo 3, el nodo 150 abrió una conexión con éste y le envía un mensaje Ping, el cual fue propagado según el mismo comportamiento anterior.

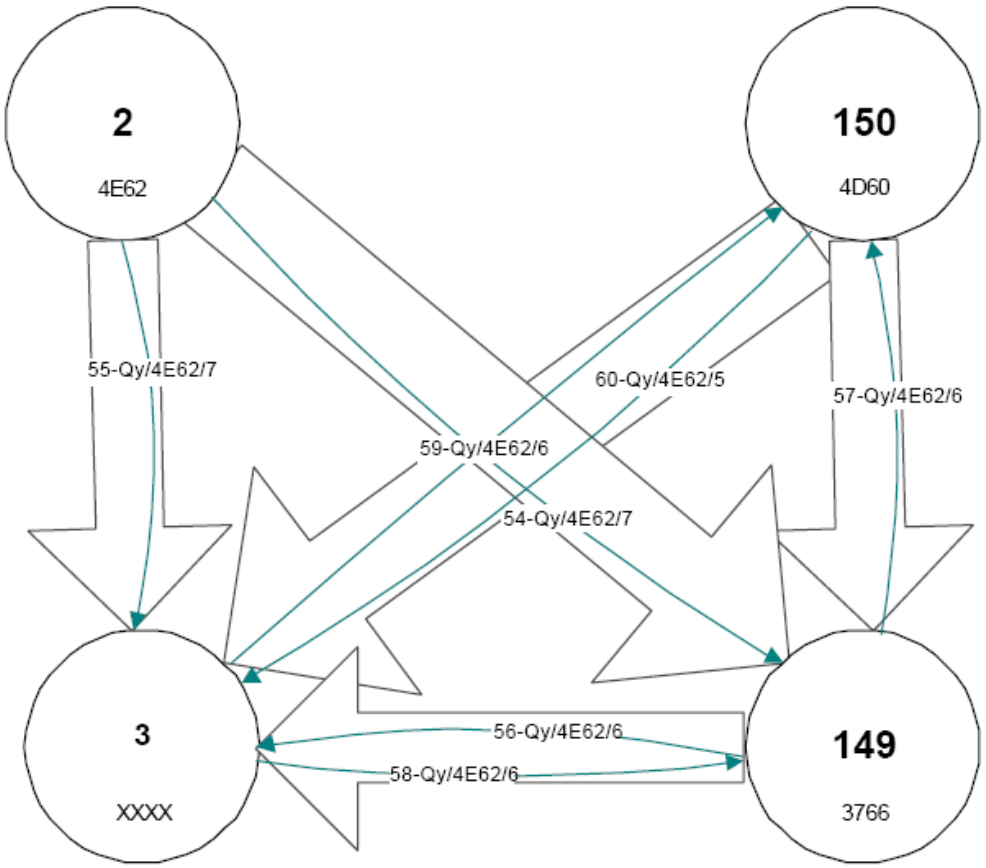


En este momento, la red Gnutella constaba de cuatro nodos, sobre la cual se ejecutaron las siguientes consultas:

En el nodo 2 se solicitó la consulta por el patrón de caracteres "*.a". Este nodo envió la consulta por todas sus conexiones abiertas (en este caso con los nodos 149 y 3). Los nodos 149 y 3 – a su vez – propagaron la consulta a través de todas sus conexiones (excepto por donde entró). Aquellos nodos que poseían archivos cuyo nombre correspondía al patrón de búsqueda (en este caso los nodos 2, 149 y 150) respondieron a la consulta enviando un mensaje por el mismo camino de la misma. Nótese que el comportamiento de los nodos en cuanto a la propagación de una consulta es similar a la de un mensaje Ping, ocurriendo lo mismo con las respuestas (respecto de los mensajes Pong).



Finalmente, en el nodo 2, se solicitó la consulta por el patrón de caracteres "i.a". Este nodo envió la consulta de la misma forma que la anterior. Los nodos 149 y 3 propagaron la misma. Ninguno de los nodos poseía archivos cuyo nombre correspondía al patrón de búsqueda, por lo tanto no la respondieron.



3.8.2.8. Conclusiones

El uso masivo del protocolo en implementaciones sobre diferentes plataformas ha demostrado que Gnutella es un punto de inicio válido para el desarrollo de servicios distribuidos basados en la filosofía compañero a compañero.

Debido a su génesis y corta existencia, Gnutella es aún un protocolo inmaduro, sometido a estudios para evaluar mejoras en eficiencia y funcionalidad (para implementar en versiones posteriores).

Como característica interesante, presenta la definición de una red a nivel aplicación, orientada al cooperativismo, que brinda elementos para favorecer la heterogeneidad, la estabilidad, la redundancia y la tolerancia a fallas.

El modelo de propagación puede servir para generar aplicaciones distribuidas que no solamente se limiten a tareas de compartir archivos. Por ejemplo, se podría utilizar este modelo, con modificaciones menores, con el objetivo de lograr una red de procesamiento distribuido basada en el cooperativismo.

4. TECNOLOGÍAS XML

Sería interesante estudiar qué resultado se obtendría de la combinación de la escalabilidad y potencia del modelo *peer-to-peer* con los modelos de negocio basados en XML.

En los últimos años se han ido abandonando los sistemas tradicionales de intercambio de datos electrónicos en favor de nuevas aplicaciones que usan tecnologías mucho más flexibles, baratas y eficientes, como son las tecnologías XML.

En este apartado trataremos de entrar en detalle en el lenguaje XML y en las tecnologías relacionadas como DOM y XSL. También se estudian las APIs de Java relacionadas con esto: JAXP yJDOM.

4.1. XML

4.1.1. Introducción

4.1.1.1. Qué es XML

El Lenguaje Extensible de Marcas, abreviado XML, describe una clase de objetos de datos llamados documentos XML y define parcialmente el comportamiento de los programas de computadora que los procesan. XML es un "perfil de aplicación" o una forma restringida de SGML, el Lenguaje Estándar Generalizado de Marcas descrito en la norma ISO 8879.

La versión 1.0 del lenguaje XML es una recomendación del W3C desde Febrero de 1998. Se puede considerar a XML como un metalenguaje, es decir, un lenguaje para crear otros lenguajes.

XML, con todas las tecnologías relacionadas, representa una manera distinta de hacer las cosas, más avanzada, cuya principal novedad consiste en permitir compartir los datos con los que se trabaja a todos los niveles, por todas las aplicaciones y soportes. Así pues, el XML juega un papel clave en este mundo actual, que tiende a la globalización y la compatibilidad entre los sistemas, ya que es la tecnología que permitirá compartir la información de una manera segura, fiable y fácil. Además, XML permite al programador y los soportes dedicar sus esfuerzos a las tareas importantes cuando trabaja con los datos, ya que algunas tareas tediosas como la validación de estos o el recorrido de las estructuras corre a cargo del lenguaje y está

especificado por el estándar, de modo que el programador no tiene que preocuparse de ello.

XML despliega principalmente su enorme potencial en el mundo de Internet y el de los negocios electrónicos, ya que existen muchos sistemas distintos que tienen que comunicarse entre sí, aunque es perfectamente aplicable a cualquier rama.

4.1.1.2. Objetivos de XML

XML fue diseñado para suplir la carencia de una tecnología similar, con los siguientes objetivos como meta:

- Que fuera idéntico al HTML a la hora de servir, recibir y procesar la información, para aprovechar toda la tecnología implantada para este último.

- Que fuera formal y conciso desde el punto de vista de los datos y la manera de guardarlos.

- Que fuera extensible, para que se pueda utilizar en todos los campos del conocimiento.

- Que fuese fácil de leer y editar, incluso para los humanos.

- Que fuese fácil de implantar, programar y aplicar a los distintos sistemas.

4.1.2. Aspectos básicos

4.1.2.1. Tipos de documentos

Existen dos tipos de documentos XML:

- **Bien formados**: son todos los que cumplen las especificaciones del lenguaje respecto a las reglas sintácticas propias de XML, sin estar sujetos a unos elementos fijados en una "gramática". Los documentos XML tienen una estructura jerárquica muy estricta, y los documentos bien formados deben cumplirla.

- **Válidos**: Además de estar bien formados, siguen una estructura y una semántica determinada por una gramática: sus elementos, la estructura jerárquica y los atributos, deben ajustarse a lo que la gramática dicte.

Se deduce fácilmente que todo documento válido es implícitamente bien formado, pero no tiene porqué cumplirse lo contrario.

4.1.2.2. Metalenguajes para definir XML

Para especificar la anteriormente citada "gramática", XML define actualmente dos metalenguajes:

- **Definición de tipos de documento (DTD)**: es una definición de los elementos que puede haber en el documento XML, y las relaciones entre ellos, sus atributos, posibles valores, etc. Cuando se procesa cualquier infomación formateada mediante XML, lo primero es comprobar si está bien formada, y luego, si incluye referencia a un DTD, se comprueba que sigue sus reglas gramaticales. La DTD puede aparecer de dos maneras:
  - Incluida en el propio documento XML.
  - Como archivo independiente, en formato XML.
- Las DTD usan una sintaxis distinta a XML y, en principio, son muy complicados de crear.
- **Esquemas (Schema XML)**: los Esquemas describen la estructura de la información (al igual que las DTDs). Han aparecido recientemente para solventar los problemas de las DTD. Un esquema XML es algo similar a un DTD, es decir, define qué elementos puede contener un documento XML, cómo están organizados y que atributos y de qué tipo pueden tener sus elementos. Están escritos en sintaxis XML, por lo que no hay que aprender un nuevo metalenguaje. Sin embargo, es más complicado realizar una correcta especificación que en DTD.

Un detalle importante de señalar a la hora de hablar de los DTD o

XML Schema es que estos lenguajes también permiten comprobar la integridad de los datos en cualquier momento. Se calcula que un 70% de las líneas de código que escribe un programador están orientadas a comprobar la integridad de los datos, es decir, comprobar si donde se supone que hay un número efectivamente lo hay, si el número es entero o cualquier otra comprobación. Los metalenguajes de XML sirven para tomar un documento XML y comprobar que los datos que en él se incluyen son válidos, comprobando si los que tenemos en el XML concuerdan con los que se deberían tener. Esta operación se puede realizar al leer el documento, liberando de esa carga al programador.

4.1.2.3. Procesado de XML

Los *parsers* son aplicaciones que procesan documentos XML. El procesador XML o procesador de XML es la herramienta principal de cualquier aplicación XML. Aparte del procesado en sí, un procesador XML también comprueba si los documentos están bien formados o válidos. La gran ventaja de usar estos procesadores XML es que se pueden incorporar a las aplicaciones, de manera que éstas manipulen y trabajen con documentos XML directamente, como suele suceder en los navegadores, que incorporan sus propios procesadores XML.

Se puede dividir a los procesadores XML en dos grupos principales:

- **sin validación**: el procesador XML no valida el documento utilizando un DTD o esquema, sino que sólo chequea que el documento esté bien formado de acuerdo a las reglas de sintaxis de XML (sólo hay una etiqueta raíz, las etiquetas están cerradas, etc).

- **con validación**: además de comprobar que el documento está bien formado según las reglas anteriores, comprueba el documento utilizando un DTD o esquema (ya sea interno o externo).

En el caso de utilizar un DTD, es preferible utilizar un procesador XML con validación.

Existen gran cantidad de procesadores XML, muchos de ellos gratuitos. Entre ellos destacan XP, Xerces, Piccolo, etc.

4.1.3. Sintaxis básica

4.1.3.1. Estructura general

A continuación se muestra un ejemplo sencillo de documento XML:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!DOCTYPE mensaje [
   <!ELEMENT mensaje (#PCDATA)>
]>
<!-- este es un comentario -->
<mensaje>Hello, world!</mensaje>
```

Antes de detallar más a fondo la sintaxis, conviene resaltar ciertos aspectos:

- Los documentos XML son sensibles a mayúsculas, esto es, en ellos se diferencian las mayúsculas de las minúsculas.
- Todos los espacios y retornos de carro se tienen en cuenta.
- Los valores de los atributos de todas las etiquetas deben ir siempre entrecomillados. Son válidas las dobles comillas (") y la comilla simple (').

Un documento XML está compuesto de marcas y datos mezclados:

- Las marcas se componen de etiquetas de inicio y de final, entidades, comentarios, delimitadores de CDATA, DTD e instrucciones de procesado.
- Todo lo demás son datos.

En los siguientes apartados, se desarrollarán las marcas.

4.1.3.2. Sintaxis de los elementos

Un documento XML está compuesto por elementos, y en su sintaxis éstos se nombran mediante etiquetas. Cada elemento puede contener a su vez otros elementos o entidades. Las entidades funcionan como abreviaciones. Los elementos pueden contener atributos (propiedades) que indican información sobre como el elemento debe ser procesado.

Hay dos tipos de elementos: los vacíos y los no vacíos. Hay varias consideraciones importantes a tener en cuenta al respecto:

- Toda etiqueta no vacía debe tener una etiqueta de cerrado: `<etiqueta>` debe estar seguida de `</etiqueta>`.

- Todos los elementos deben estar perfectamente anidados: no es válido poner:

- <ficha><nombre>Josef</ficha></nombre>

- y sí lo es sin embargo:

- <ficha><nombre>Josef</nombre></ficha>

- Los elementos vacios son aquellos que no tienen contenido dentro del documento. La sintaxis correcta para estos elementos implica que la etiqueta tenga siempre esta forma: `<etiqueta/>`.

4.1.3.3. Prólogo y Declaración de un documento

Los documentos en XML se inician con una declaración de XML especificando:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

- **version**: indica la versión de XML usada en el documento. La actual es la versión 1.0. Es obligatorio ponerlo, a no ser que sea un documento externo a otro que ya lo incluía.

- **encoding**: la forma en que se ha codificado el documento.

Se puede poner cualquiera, y depende del procesador XML el entender o no la codificación. Por defecto es UTF-8.

- **standalone**: indica si el documento va acompañado de un DTD ("no"), o no lo necesita ("yes"). En principio no hay porqué ponerlo, porque luego se indica el DTD si se necesita.

Deben ser seguidos de la declaración de su DTD (document type declaration)

```
<!DOCTYPE mensaje [   <!ELEMENT mensaje (#PCDATA)>    ]>
```

La declaración de su DTD puede hacerse externa al documento:

```
<!DOCTYPE mensaje SYSTEM "mensaje.dtd">
```

4.1.3.4. Comentarios

Los comentarios pueden ocurrir en cualquier lugar fuera de una marca:

```
  <!-- comentario -->
```

4.1.3.5. Declaraciones de Elementos

Ayuda a restringir el contenido de un elemento. Algunos ejemplos son:

```
<!ELEMENT br          EMTPY>

<!ELEMENT p          (#PCDATA|emph)*>

<!ELEMENT container   ANY>
```

4.1.3.6. Definiciones de Atributos

Los atributos especifican características especiales que aplican a un elemento en particular. Se declaran:

```
<!ATTLIST subject form (bold|italic|normal) "normal">

...

<subject form="italic">...</subject>
```

```
<subject>...</subject>
```

4.1.3.7. Entidades

Las entidades sirven dos propósitos:

- Abreviaturas
- Para integrar caracteres no incluidos en el conjunto de caracteres.

Un ejemplo:

```
<!ENTITY eacute CDATA "&#233;">

<!ENTITY buap "Benem&eacute;rita Universidad Aut&oacute;noma de Sevilla">

...

&buap;
```

4.1.3.8. Instrucciones de Procesado

No son parte del documento sino que ofrecen al procesador información adicional:

```
<?xml version="1.0"?>
```

4.1.3.9. Secciones CDATA

Permiten integrar texto en un documento en XML que de otra forma sería interpretado como etiquetas:

```
<![CDATA[<poema>This is the sea</poema>]]>
```

4.1.4. Tecnologías XML

Debido a la universalidad de XML, se han desarrollado numerosas tecnologías y soluciones utilizando toda la potencia que XML ofrece. Se han creado numerosos lenguajes específicos a partir de XML para suplir la falta de estándares en muchas áreas, o incluso para sustituir los obsoletos y propietarios. El resultado es una nueva concepción de expresar los datos y de enfocar las comunicaciones en la red.

Resultaría imposible describir y mencionar la totalidad de las tecnologías relacionadas con XML. Cada día, se añaden a la lista nuevas soluciones. Se citan a continuación varias tecnologías de distintos campos, para comprobar a manera de ejemplo el alcance y eficacia de XML:

- SMIL: Lenguaje Sincronizado de Integración Multimedia.
- MathML: Lenguaje de Marcas Matemático.
- SVG: Gráficos de Vector Escalables.
- DrawML: Lenguaje de Meta Dibujo Estándards eCommerce.
- cXML: XML para comercio.
- ebXML: XML para negocios electrónicos.
- DMTF: Manejo Distribuido de Tareas Forzadas.

No se han mencionado tecnologías y productos que usan XML para realizar sus transacciones o autodefinirse, como puede ser JXTA, WSDL, etc, porque la lista sería interminable.

4.2. DOM

4.2.1. Qué es DOM

El Modelo de Objetos del Documento (DOM) es una interfaz de programación de aplicaciones (API) para documentos HTML y XML. Define la estructura lógica de los documentos y el modo en que se accede y manipula un documento. En la especificación del DOM, el término "documento" se utiliza en un sentido amplio, ya que XML presenta los datos como documentos, más que como datos en sí.

Con DOM, los programadores pueden construir documentos, navegar por su estructura y añadir, modificar o eliminar elementos y contenido. Se puede acceder a cualquier cosa que se encuentre en un documento HTML o XML, y se puede modificar, eliminar o añadir usando DOM, salvo algunas excepciones.

DOM es una especificación del W3C, por lo que uno de sus objetivos importantes es proporcionar un interfaz estándar de programación que pueda utilizarse en una amplia variedad de entornos y aplicaciones. DOM se ha diseñado para ser utilizado en cualquier lenguaje de programación.

4.2.2. Aspectos básicos de DOM

El DOM es una API de programación para documentos. Guarda una gran similitud con la estructura del documento al que modeliza. En la siguiente figura se muestra una estructura HTML de tabla y su representación DOM correspondiente:

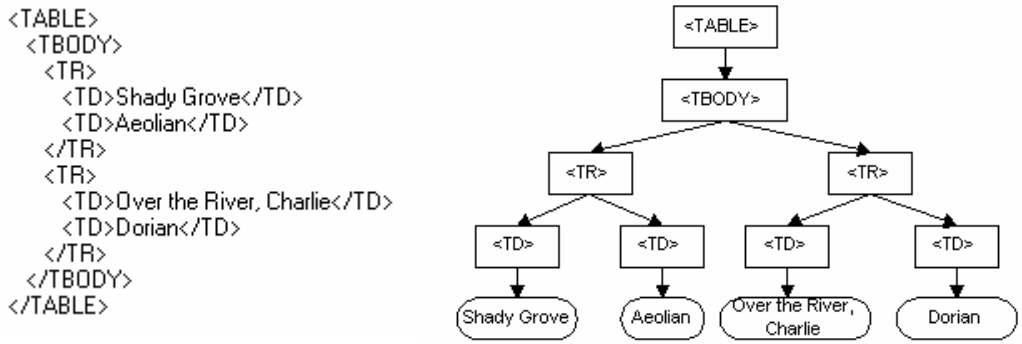


Figura 14: Representación en DOM de una estructura

En el DOM, los documentos tienen una estructura lógica muy parecida a un árbol. Sin embargo, DOM no especifica que los documentos deban ser implementados como un árbol, ni tampoco especifica cómo deben implementarse las relaciones entre objetos. El DOM es un modelo lógico que puede implementarse de cualquier manera que sea conveniente. En esta especificación, se usa el término modelo de estructura para describir la representación en forma de árbol de un documento evitando confusiones con implementaciones.

4.2.3. Niveles de DOM

DOM se ha dividido en varios niveles, descritos en distintas recomendaciones. Han sido publicados secuencialmente en el tiempo, y cada nuevo nivel soporta y amplia las capacidades del anterior, solventando problemas y definiendo nuevas capacidades. Hasta la fecha, W3C ha publicado 3 niveles:

- **Nivel 1**: Como modelo de objetos, el DOM identifica:

- las interfaces y objetos usados para representar y manipular un documento

- la semántica de estas interfaces y objetos, incluyendo comportamiento y atributos

- las relaciones y colaboraciones entre estas interfaces y objetos.

DOM Nivel 1 se compone actualmente de dos partes: el Núcleo del DOM y el DOM HTML. El Núcleo del DOM representa la funcionalidad usada para los documentos XML, y también sirve de base para el DOM HTML. El núcleo define un conjunto mínimo de objetos e interfaces para acceder y manipular objetos del documento, y es suficiente para permitir a los desarrolladores de programas y a los autores de scripts web acceder y manipular contenido HTML y XML analizado dentro de productos conformes.

El Nivel 1 es bastante limitado, y deja al aire muchas cuestiones, como:

- Validación contra un esquema.

- Control para representar documentos por medio de hojas de estilo.

- Control de acceso.

- Seguridad de hilos de proceso o threads.

- Eventos.

- **Nivel 2:** Mejora varias cuestiones del Nivel 1, aunque no todas. En particular, proporciona soporte para los espacios de nombres, puede crearse documentos DOM desde cero, importar nodos de otros documentos DOM, etc.

- **Nivel 3:** Proporciona soporte a estándares recientes, como los esquemas XML, y además proporciona nuevas

características como la posibilidad de comparar nodos, importarlos directamente de otro DOM, normalización de documentos, etc.

4.3. XSL

XSL (*XML Stylesheet Language*, Lenguaje de hoja de estilos XML) es una familia de recomendaciones W3C para la definición de transformaciones y formas de presentar documentos XML mediante hojas de estilo.

Una hoja de estilo es un archivo que describe cómo mostrar un documento XML de un determinado tipo. Es comparable a la labor de una CSS (Cascading StyleSheets), pero no son compatibles entre sí. Las hojas de estilo XSL sí son compatibles con CSS2 y comparten funcionalidad, aunque están descritas en sintaxis distintas.

La familia de recomendaciones XSL consiste en tres partes fundamentales:

- **XSLT** (*XSL Transformations*, Transformaciones XSL): define un lenguaje de transformación de XML en otros formatos.

- **XSL-FO** (*XSL Formatting Objects*, Objetos de Formateo XSL): define un vocabulario XML para especificar semánticas de formateo para formatos binarios.

- **XPATH** (*XML Path Language*, Lenguaje de ruta XML): un lenguaje de expresiones usado por XSLT para acceder o referenciar determinadas partes de un documento XML.

La versión actual publicada por el W3C es la 1.0. Existe en estos momentos un borrador de la versión 2.0. En este apartado se discutirá sólo la versión 1.0.

4.3.1. XSLT

4.3.1.1. Qué es XSLT

Especifica el lenguaje para realizar transformaciones desde XML a otros formatos como HTML, otro XML, texto plano, etc. Una transformación XSLT puede realizar las siguientes operaciones:

- Convertir documentos XML en otros documentos XML.

- Convertir un XML que esté regido por un determinado DTD a otro que obedezca otro diferente.

- Convertir un documento XML sólo bien-formado a otro que siga un DTD determinado.

- Convertir un documento XML a otros formatos como WML, HTML, etc.

Además, XSLT también puede servir como lenguaje de programación, pero su uso principal es la transformación de XML.

4.3.1.2. Ventajas y desventajas

XSLT es una herramienta muy poderosa, pero hay que pagar un precio por sus posibilidades. Entre las ventajas se encuentran:

- La salida no tiene por qué ser HTML para visualización en un navegador, sino que puede estar en muchos formatos.

- Permite manipular de muy diversas maneras un documento XML: reordenar elementos, filtrar, añadir, borrar, etc.

- Permite acceder a todo el documento XML, no sólo al contenido de los elementos.

- XSLT es un lenguaje XML.

Como desventajas:

- Su utilización es más compleja y algo confusa, aunque existen herramientas visuales que facilitan enormemente esta tarea.

- Consume cierta memoria y capacidad de proceso, pues se construye un árbol con el contenido del documento.

La conveniencia de usar XSLT dependerá de la aplicación.

4.3.1.3. Formas de uso

XSLT admite varias formas de empleo. Entre ellas destacan:

- Visualizar directamente en un navegador el documento XML que tiene asociada una hoja XSLT. El navegador debe tener incorporado un procesador XSLT.

- Ejecutar el procesador XSLT independientemente del navegador. Se debe suministrar las entradas necesarias (fichero origen y hoja XSLT a utilizar) y el procesador generaría la salida en un fichero.

- Realizar las transformaciones dentro de un programa en un servidor y enviar a los clientes sólo el resultado de la transformación.

4.3.1.4. La transformación XSLT

La siguiente figura describe el proceso de transformación mediante XSLT:

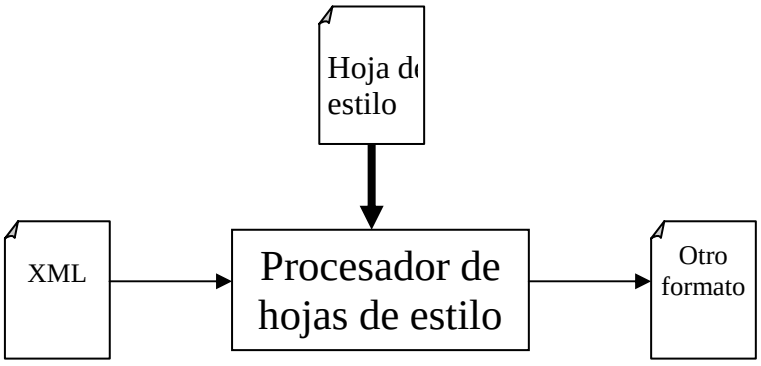


Figura 15: Transformación con XSLT

La transformación XSLT en sí se denomina hoja de estilo. Suele estar implementado en un archivo de texto, y escrito en XML. El procesador de hojas de estilo realiza la transformación XSLT de un archivo XML a otro de distinto formato, usando para ello las reglas contenidas en la hoja de estilo especificada.

Una transformación expresada en XSLT describe reglas para tranformar un árbol origen en un árbol resultado. La transformación es realizada mediante la asociación de patrones con plantillas. Un determinado patrón es comparado con elementos del árbol origen, mientras que las plantillas son instanciadas para crear una parte del árbol resultado. A cada elemento del árbol origen que coincide con un patrón determinado se realiza una transformación basada en la plantilla asociada a dicho patrón. En general, la estructura resultante será completamente distinta de la de origen, ya que los elementos pueden ser eliminados, reordenados, reconvertidos, copiados o incluso se pueden añadir nuevas estructuras arbitrarias.

La hoja de estilo XSLT contiene las reglas de transformación, y cada regla está dividida según el esquema anteriormente descrito de patrones y plantillas asociadas. Esto permite que sea aplicable a una amplia variedad de documentos que tengan estructuras origen similares. Cuando se encuentra un elemento que cumple un patrón, se instancia una plantilla, que puede contener tanto una estructura literal de elementos de resultado como elementos del espacio de nombre XSLT con instrucciones para crear elementos del árbol resultado. Al instanciar la plantilla, se ejecuta cada instrucción y se emplaza como fragmento del árbol resultado que se ha creado. Las instrucciones pueden procesar elementos hijos de la misma manera: encontrando el patrón apropiado e instanciando la plantilla. Si se encuentra un elemento que coincide con un patrón que tiene asociado más de una plantilla, sólo se ejecuta una de ellas.

Los procesadores de hojas de estilo son aplicaciones, normalmente en conjunción con un procesador XML, que realizan la transformación XSLT, buscando elementos que cumplan con un patrón y aplicándoles la plantilla asociada.

XSLT usa el lenguaje de expresión definido por XPath para las siguientes tareas:

- Selección de elementos para procesarlos.
- Procesamiento condicional.
- Generación de texto.

4.3.1.5. Sintaxis XSLT

La hoja de estilo está definida en un fichero XML. La sintaxis básica de XSLT consiste en:

- Elemento raíz: siempre debe ser el siguiente, especificando la versión y el espacio de nombres que se va a usar:

```
      <xsl:stylesheet  version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

- `xsl:output` : controla qué tipo de salida va a producir la hoja XSL. Lo normal es producir XML, aunque también se puede producir HTML o texto plano.

- `xsl:template` : define una transformación. Tiene un atributo "`match`" que dice a qué elementos se aplica la transformación.

- `xsl:apply-templates` : aplica los `xsl:template` asociados a cada elemento que haya en la parte del documento XML que se esté procesando.

- `xsl:value-of` : obtiene el valor de un nodo.

- `xsl:text` : escribe texto de forma literal en la salida.

- `xsl:for-each` : itera sobre un conjunto de elementos.

- `xsl:when`, `xsl:choose` y `xsl:otherwise` : sirven para expresar condiciones multiples.

- `xsl:if` : se procesa si se cumple una condición.

4.3.1.6. Herramientas XSLT

A menudo se usa XSLT dentro de un entorno de publicación como el Cocoon, o el IBM Transcoding Publisher, AxKit, etc. Un entorno de publicación de XML permite mantener sitios completos basados en XML, y generar páginas en diferentes formatos a partir de ellos. Cocoon es

una herramienta gratuita y Open Source basada en Java, y además una de las más avanzadas en el sector. IBM TP, por el contrario, es propietario y de elevado coste, pero tiene dos grandes ventajas: forma parte del servidor de aplicaciones WebSphere y cuenta con interfaz gráfico avanzado.

En la mayoría de los casos, se usa un procesador de estilo para aplicar las hojas de estilo dentro de una aplicación, o usarlas desde línea de comandos a partir de otra aplicación u otro lenguaje de programación. Los procesadores más conocidos son:

- Xalan: es gratuito, muy usado y está escrito en Java. Necesita un procesador XML para funcionar, como el Xerces, aunque suele venir incluido en la distribución. La versión actual implementa XSL v1.0 y XPath v1.0.

- Saxon: un procesador bastante rápido, también escrito en Java, con su propio *parser* incluido, aunque se puede usar uno externo.

- Otros como el XT de James Clark, o el Sablotron.

La creación de hojas de estilo puede ser costosa al principio. Existen herramientas visuales para su desarrollo, además de que pemiten editar XML y aplicar las transformaciones directamente. Entre ellas se encuentran XMLSpy y eXcelon Stylus, aunque son propietarias.

4.3.2. XSL-FO

XSL-FO es un vocabulario para especificar semánticas de formateo. Su utilidad es la presentación de documentos, debido a que :

- XML no da información de la presentación, sólo del contenido.

- XSLT no dice la presentación, sino que realiza una transformación para así hacer el XML presentable, por lo tanto necesita un archivo origen y una hoja de estilos.

XSL-FO resuelve estos problemas: incluye en el XML la información necesaria para poder presentar el documento. Esta información de

formateo son los objetos de formateo. XSL-FO proporciona una serie de marcas que pueden ser usadas para definir cómo un documento será representado de cara al usuario, como el diseño general, tipo de fuente, colores, y otras propiedades del diseño.

La W3C ha definido una enorme cantidad de objetos de formateo. No sólo definen objetos para documentos impresos, también deja la puerta abierta para contenidos multimedia.

XSL-FO es muy útil para los documentos XML que deben ser leídos por seres humanos; en principio, no aporta nada a aquellos que actúan como datos intercambiados y usados exclusivamente por aplicaciones.

La gran ventaja de XSL-FO es que, al ser documentos XML con marcas FO, son portables a cualquier plataforma, a diferencia de la gran cantidad de formatos propietarios existentes. El uso más extendido de los FO es la representación de XML en el formato PDF de Adobe.

4.3.3. XPath

4.3.3.1. Objetivos de XPath

XPath proporciona una sintaxis y semántica comunes para funcionalidades compartidas entre XSLT y XPointer. El objetivo principal de XPath es direccionar partes de un documento XML. Como soporte para este objetivo principal, también proporciona facilidades básicas para manipulación de cadenas, números y booleanos. XPath utiliza una sintaxis compacta y no-XML para facilitar el uso de XPath dentro de URIs y de valores de atributos XML. XPath opera sobre la estructura lógica abstracta de un documento XML, más que en su sintaxis superficial. XPath obtiene su denominación por el uso que hace de una notación de caminos, como en los URLs, para navegar a través de la estructura jerárquica de un documento XML.

Además de su uso para direccionar, XPath está diseñado también de modo que tiene un subconjunto natural que puede usarse para cotejar (comprobar si un nodo encaja o no con un patrón); este uso de XPath está descrito en XSLT, y se usa para la comprobación de los elementos del XML origen con los patrones de las reglas de transformación definidos en la hoja de estilo.

4.3.3.2. Aspectos básicos

XPath modela un documento XML como un árbol de nodos. Hay diferentes tipos de nodos, incluyendo nodos elemento, nodos atributo y nodos texto. XPath define un modo de calcular un valor de cadena para cada tipo de nodo.

La construcción sintáctica básica en XPath es la expresión. Las expresiones son evaluadas para producir un objeto, que tendrá uno de los siguientes cuatro tipos básicos:

- conjunto de nodos (una colección desordenada de nodos sin duplicados) .
- booleano (verdadero o falso).
- número (un número en punto flotante) .
- cadena de texto (secuencia de caracteres).

Las expresiones se analizan dividiendo la cadena de caracteres a analizar en *tokens* y a continuación analizando la secuencia de *tokens* resultante.

Un tipo importante de expresión es el camino de localización. Un camino de localización selecciona un conjunto de nodos relativo al nodo de contexto. Los caminos de localización pueden contener recursivamente expresiones utilizadas para filtrar conjuntos de nodos.

La especificación que describe XPath es larga y compleja, por lo que no se entrará en más detalle aquí. La mejor manera de entender y utilizar XPath es viendo ejemplos.

4.3.3.3. Algunos ejemplos básicos

Las reglas básicas de las expresiones XPath son:

- // : representa el nodo raíz del documento.
- / : desciende un nivel en la jerarquía.

- : representa todos los elementos contenidos en la ruta.

- [i] : selecciona la i-ésima ocurrencia del elemento seleccionado en la ruta.

- @atributo : selecciona todos los atributos llamados "atributo" en la ruta.

Además, existe un conjunto de operadores ya definidos como *last*, *count()*, ... También se permiten operadores booleanos.

Para este ejemplo se usará una estructura XML imaginaria, representada a continuación:

```
<AAA>
       <BBB id="x"/>
           <CCC/>
       <BBB/>
       <DDD>
              <EEE/>
              <FFF/>
```

Los resultados de aplicar las siguientes expresiones son:

Expresión XPath	Objetos devueltos
/AAA/BBB	<BBB id="x"/> <BBB/>
//CCC	<CCC/>
/*/*/EEE	<EEE/> <EEE/>

/AAA/BBB[1]	<BBB id="x"/>
/AAA/BBB[last()]	<BBB/>
//BBB[@id]	<BBB id="x"/>
//@id	id="x"

4.4. XML Y JAVA

Las aplicaciones actuales, como servicios web, aplicaciones web, etc. dependen de la capacidad de los participantes de poder comunicarse aunque usen distintos sistemas de información. XML es una tecnología clave para resolver este problema, puesto que define una nueva manera de interoperabilidad de información entre sistemas muy dispares.

Asímismo, las aplicaciones también dependen de la capacidad de interoperabilidad entre plataformas completamente distintas. La solución aquí es Java, ya que es independiente de la plataforma y posee código portable.

Por estas dos razones, es evidente que Java y XML juntos constituyen una poderosa herramienta de desarrollo, obteniendo la independencia de plataforma y de sistema. De esta manera, han surgido recientemente multitud de APIs para el tratamiento de XML. Entre las opciones más destacables están:

- **JAXP** (*Java API for XML Processing*) : es la API referencia para el manejo básico de XML proporcionada por Sun Microsystems. Aglutina a 3 APIs:
  - SAX: para procesado general de XML.
  - DOM: para procesado bajo la especificación DOM de documentos XML.
  - XSLT: para realizar las transformaciones XSLT.
- Es muy completa, y con ella se puede hacer prácticamente de todo.

- **JAXB** (*Java Architecture for XML Binding*) : genera clases Java a partir de esquemas XML. La proporciona Sun.

- **JAXR** (*Java API for XML Registries*) : proporciona formas de acceder a registros de negocios en Internet. También está proporcionada por Sun.

- **JAX-RPC** (*Java API for XML-Based RPC*): implementación de RPC basado en SOAP. También está proporcionada por Sun.

- **JAXM** (*Java API for XML Messaging*) : interfaz para el intercambio de mensajes y establecimiento de transacciones de negocios. Está basado en SOAP.

- **dom4j**: es un marco de trabajo para el manejo de XML basado en DOM y SAX. Es un proyecto de software libre.

- **JDOM**: librerías para manipular DOM optimizado para Java.

- **Xerces**: procesador XML, está muy extendido su uso. Pertenece al Apache XML Project.

- **Xalan**: procesador de hojas de estilo XSLT, también pertenece al Apache XML Project.

Se ha citado aquí sólo una pequeñísima parte del conjunto de software desarrollado para el manejo de XML en Java, aunque son generalmente los más usados.

En los siguientes apartados, se comentan más profundamente las dos APIs elegidas para la realización de este proyecto.

4.5. JAXP

Ha sido desarrollada por Sun, y actualmente forma parte del paquete JWSDP (*Java Web Services Development Pack*). Constituye una referencia para otros proyectos similares, que están basados a menudo en las implementaciones de SAX y DOM dados por JAXP.

JAXP incluye los estándares de proceso SAX (*Simple API for XML Parsing*) y DOM (*Document Object Model*), de forma que se puede elegir la forma de procesar los datos XML: como un flujo de eventos o como una representación en objetos. SAX y DOM están definidos por el grupo XML-DEV y por la W3C respectivamente. Además, JAXP soporta el estándar XSLT, permitiendo el control sobre la presentación de los datos y la conversión a otros formatos como HTML. JAXP permite los espacios de nombre, permitiendo evitar conflictos de nombrado con varias DTDs.

JAXP ha sido diseñado para ser flexible, por lo que permite usar cualquier procesador que cumpla con XML y cualquier procesador XSL.

Los principales paquetes de JAXP son:

- **`javax.xml.parsers`**: contiene las clases factoría indispensables para manejar XML:
  - `SAXParserFactory`: proporciona un objeto `SAXParser`, que implementa un procesador para XML.
  - `DocumentBuilderFactory`: proporciona un `DocumentBuilder`, que construye objetos `Document` que cumplen con el estándar DOM.
- **`org.w3c.dom`**: define la clase `Document` de DOM y las clases que representan los elementos de un DOM.
- **`org.xml.sax`**: define las APIs básicas de SAX.
- **`javax.xml.transform`**: proporciona la API para las transformaciones XSLT.

4.5.1. SAX

SAX es un mecanismo de acceso serie y por eventos, que realiza procesamiento elemento a elemento. El API para este nivel lee y escribe XML a un repositorio de datos o a la web. Este API es realmente un producto de colaboración de la lista de correo XML-DEV, en vez de un producto del W3C, pero tiene las mismas características que una recomendación de W3C.

Los paquetes que integran SAX son:

- **`org.xml.sax`**: define los interfaces SAX de manejo de eventos.

- **`org.xml.sax.helpers`**: para crear analizadores distintos a los proporcionados por la implementación de Sun.

- **`javax.xml.parsers`**: para crear el analizador de la implementación de Sun.

- **`com.sun.xml.parser`**: contiene los analizadores de la implementación, con y sin validación frente a un DTD.

SAX está dirigido por eventos, de forma que su funcionamiento es, en líneas generales:

- Se crea un analizador `SAXParser` con la factoría `SAXParserFactory`.

- Se registra el manejador usado por la aplicación con un analizador SAX.

- El analizador procesa el XML secuencialmente, ya que SAX sólo implementa acceso serie.

- Cada vez que ocurre un determinado evento, el analizador invoca una retrollamada al manejador, que tendrá implementado un determinado comportamiento ante dichos eventos. Las interfaces `DocumentHandler`, `ErrorHandler`, `DTDHandler`, y `EntityResolver` definen los métodos que son llamados ante la ocurrencia de un evento, y deben ser implementados por el manejador.

La siguiente figura muestra el esquema seguido por SAX:

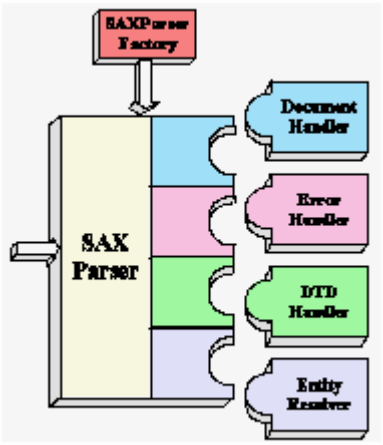


Figura 16: Analizador SAX

Debido a su característica de acceso serie, no permite acceso aleatorio, por lo que no es útil para aplicaciones de que necesiten una modificación compleja de la estructura del documento; sin embargo, y debido precisamente a esto, es muy útil para aplicaciones del lado del servidor, aplicaciones que requieran alto rendimiento o programas que no necesiten una representación del XML en memoria de forma permanente.

4.5.2. DOM

El API DOM, definido por la W3C, es generalmente más fácil de usar. Proporciona una relativamente familiar estructura de árbol de objetos, por lo que el acceso a la estructura XML es aleatorio. Los paquetes que integran esta API son:

- **`org.w3c.dom`**: Define los interfaces DOM para documentos XML según lo especificado por W3C.

- **`javax.xml.parsers`**: Define la clase `DocumentBuilderFactory` y la clase `DocumentBuilder`, que devuelve un objeto que implementa el interfaz `Document` W3C.

- **`com.sun.xml.tree`:** La implementación Java XML de Sun de las librerías DOM.

La siguiente figura muestra cómo funciona DOM:

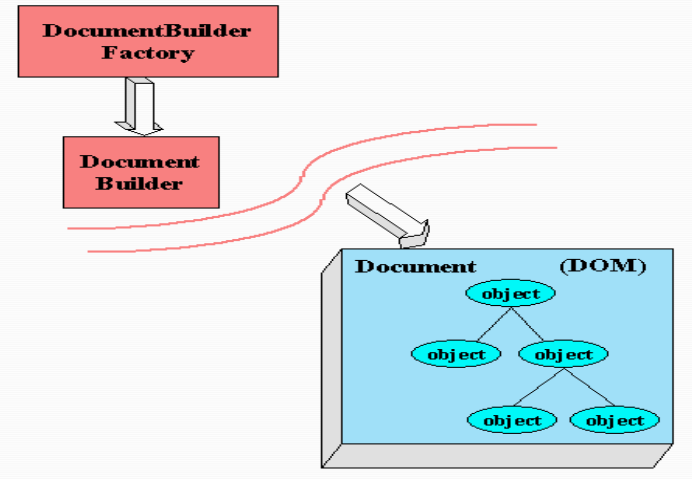


Figura 17: Modelo de API DOM

El API DOM es ideal para aplicaciones interactivas, porque el modelo de objeto completo se presenta en memoria, donde puede ser accedido y manipulado por el usuario. Sin embargo, construir el DOM requiere la lectura de toda al estructura XML y mantener el árbol de objetos en memoria, lo que significa un mayor consumo de CPU y de memoria.

Este API, aunque es relativamente sencillo de manejar, es bastante incómodo a la hora de realizar determinadas operaciones, ya que cumple con la recomendación W3C y no está optimizado para Java. Existen otras opciones a la hora de usar DOM más potentes y cómodas, algunas incluso están optimizadas específicamente para Java.

4.5.3. XSLT

Para usar la API XSLT, hay que instanciar la factoría, `TransformerFactory`, y crear un objeto `Transformer`. Este objeto actúa como procesador de hojas de estilo. Se le debe pasar un objeto fuente (un flujo, un objeto SAX o uno DOM), y devolverá un objeto resultado (un flujo, un objeto SAX o uno DOM). También se le debe dar las instrucciones de transformación en la hoja de estilo; si no se especifica hoja de estilo, la fuente se copia al resultado.

Los paquetes que componen XSLT son:

- **`javax.xml.transform`**: define las clases factoría y transformador XSLT.

- **`javax.xml.transform.dom`**: define objetos fuente y resultado basados en DOM.

- **`javax.xml.transform.sax`**: define objetos fuente y resultado basados en SAX.

- **`javax.xml.transform.stream`**: define objetos fuente y resultado basados en flujos cualesquiera.

La siguiente figura muestra el funcionamiento de esta API:

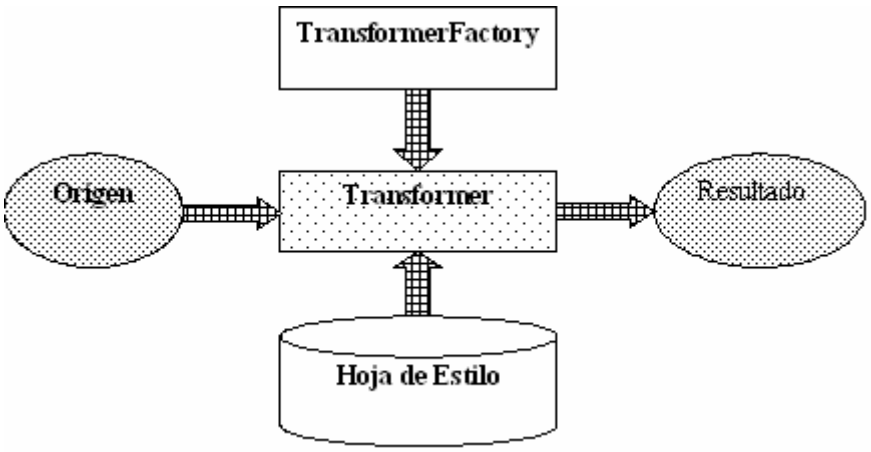


Figura 18: Modelo de API DOM

4.6. JDOM

4.6.1. Qué es JDOM

JDOM (Java DOM) es una librería de código libre para la manipulación de datos XML, y está específicamente optimizada para Java. Aunque es similar al modelo DOM del W3C, es un modelo de objeto de documento alternativo, y no está creado ni modelado sobre DOM. La principal diferencia es que DOM fue creado para ser independiente del lenguaje de programación, y orientado para manipulación de páginas HTML por parte de JavaScript, mientras que JDOM ha sido creado específicamente para Java y, por tanto, tiene las ventajas inherentes a Java, como sobrecarga de métodos, mayor eficiencia, etc.

La implementación de DOM en JAXP sigue al pie de la letra las recomendaciones de la W3C, por lo que es más complicado usar esa

API. En cambio, JDOM es más "natural" y evidente para un programador Java, además de que se ganará notablemente en rapidez, tanto en desarrollo como en ejecución. En algunos casos, operaciones que llevan 10 líneas de código en DOM se implementan con sólo 2 en JDOM. Como ejemplo para comparar, JDOM es a DOM como RMI es a CORBA: RMI está optimizado para Java mientras que CORBA es independiente del lenguaje.

JDOM es un proyecto de código abierto con licencia al estilo Apache. Ha sido desarrollado por una comunidad con sede en www.jdom.org. Además, ha sido aceptada por la JCP de Sun como la petición de especificación JSR-102, por lo que se convertirá próximamente en una especificación formal de Java. Actualmente, la versión vigente es la beta 9 y se han propuesto candidatos para la beta 10, que se prevé que sea la última antes de publicar la primera versión oficial 1.0. En este apartado se discutirá la versión beta 8, puesto que es la más reciente de la que se haya publicado un manual de usuario, y será la versión usada en este proyecto.

JDOM incluye soporte para XSLT en forma de envolturas de su clase `Document` para los flujos de entrada y salida. También implementa XPath.

4.6.2. Aspectos básicos de JDOM

Al igual que DOM, JDOM representa un documento XML como un árbol compuesto de elementos, atributos, comentarios, nodos de texto, secciones CDATA, etc. El árbol está disponible en cualquier momento, ya que se mantiene en memoria durante la ejecución. Sin embargo, todos los tipos de nodos del árbol están representados en JDOM por clases concretas de Java, y no por interfaces como en DOM de JAXP.

Al estar escrito en y para Java, todas las clases básicas de JDOM tienen los métodos `toString()` y `hashCode()`, por citar algunos, además de implementar las interfaces `Cloneable` y `Serializable`. Los nodos hijos son almacenados en instancias de la conocida clase `java.util.List`. JDOM pretende ser respetuoso tanto con XML como con Java.

JDOM no incluye un procesador XML, sino que depende de un procesador SAX. En particular, incorpora Xerces 1.4.4, aunque podría trabajar con procesadores de SAX2 como Xerces-2 o Crimson. JDOM no sólo es compatible con SAX, sino también con DOM: es posible generar un JDOM a partir de un DOM, pero en general no es muy recomendable, ya que se crearían dos estructuras arbóreas que

consumen muchos recursos. Es preferible, siempre que se pueda, construir JDOM a partir de SAX.

JDOM comprueba que los elementos estén bien formados cuando se crea un nuevo árbol desde cero. Cuando se crea un nuevo documento, ya sea desde un flujo generado por un procesador o desde cero, se carga en memoria y puede ser alterado en su totalidad, añadiendo, eliminando o moviendo elementos, siempre siguiendo las restricciones de XML. Cuando se ha terminado de trabajar con el documento en memoria, JDOM permite:

- serializarlo en disco mediante un archivo
- serializarlo como flujo de bytes.
- producir un SAX mediante eventos.
- producir un documento DOM.

4.6.3. Los paquetes JDOM

La librería JDOM consiste en seis paquetes:

- **`org.jdom`**: representa un documento XML y sus componentes: `Attribute`, `CDATA`, `Comment`, `DocType`, `Document`, `Element`, `EntityRef`, `Namespace`, `ProcessingInstruction`, y `Text`, los mismos componentes que posee XML.
- **`org.jdom.input`**: contiene clases para construir documentos JDOM. Las más importante son `SAXBuilder`, que crea un JDOM usando SAX a partir de un archivo o un flujo XML, y `DOMBuilder`, que construye desde un árbol DOM.
- **`org.jdom.output`**: contiene clases para "imprimir" un documento JDOM en un flujo de *bytes* o un archivo. La más importante es `XMLOutputter`, que posee una amplia variedad de opciones de configuración. También existen clases para "imprimir" en eventos SAX, en árboles DOM, o en árboles `JTree`.

- **`org.jdom.transform`**: contiene clases para realizar las transformaciones XSLT.

- **`org.jdom.xpath`**: contiene clases que soportan XPath.

- **`org.jdom.adapters`**: para propósitos de compatibilidad, los usuarios de JDOM no lo deben usar.

Como puede observarse el campo de las tecnologías XML es muy amplio y versátil. Dada la libertad que da al usuario se considera idóneo el empleo de este lenguaje en los ficheros que contendrán la información de los productos que se anunciarán en la red peer-to-peer.

5. CXML

La elección de cXML como lenguaje de comercio electrónico respecto a los otros estándares que se mencionaron en el apartado dedicado al Comercio Electrónico se debe a los siguientes aspectos:

- Es un **lenguaje XML**: actualmente el comercio electrónico está siendo implantado con XML, alejándose de las fórmulas tradicionales como EDI.

- Es un **estándar abierto y gratuito**: se puede usar gratuitamente, e implementar sobre cualquier plataforma. La mayoría de las opciones competidoras también eran gratuitas en principio, pero algunas exigían el empleo de *software* propietario para su desarrollo (caso de BizTalk de Microsoft).

- Es uno de los lenguajes de comercio electrónico más **sencillo**: los documentos definidos son muy parecidos a cualquier documento de negocios impreso en papel, por lo que se ajusta perfectamente a cualquier modelo de negocio que se quiera implementar, especialmente en el caso de ComercioP2P, que pretende ofrecer un conjunto sencillo y básico de transacciones comerciales.

- La **documentación es clara y precisa**: el manual de usuario es muy aclaratorio y sencillo, y aporta numerosos ejemplos de los documentos definidos. En otras opciones, este aspecto está más descuidado, tanto por exceso de documentación como por defecto.

- **Sólo define el lenguaje** de comercio, los mensajes que se van a intercambiar. Las otras opciones, como por ejemplo ebXML, definen más bien el modelo de negocio global mediante procesos que se debe seguir. Sin embargo, para ComercioP2P se deseaba únicamente un modelo de documentos a emplear, sin entrar en más complejidades que complicasen e hicieran confuso el sistema.

Pasemos a continuación a hacer una descripción más detallada de este estándar.

5.1. INTRODUCCIÓN

5.1.1. Qué es cXML

cXML (Commerce eXtensible Markup Language) es un lenguaje abierto y versátil para los requisitos de transacciones de:

- Catálogos de productos electrónicos.
- Catálogos de *punchout* cXML.
- Aplicaciones de adquisición.
- Comunidades de compra.

Está completamente basado, como su nombre indica, en XML, que le proporciona las características de universalidad y versatilidad necesarias para poder usarlo en multitud de situaciones relacionadas con el comercio electrónico.

Otras ventajas de cXML son:

- Muy sencillo de implementar.
- Variedad de documentos soportados.
- Gran aceptación.
- Deja atrás en gran medida la sintaxis de EDI.
- Es extensible y adaptable.

cXML define únicamente el lenguaje. Aparentemente, esto parece una debilidad, sin embargo constituye un punto fuerte a su favor: permite que sea portable a cualquier plataforma o sistema operativo, dejando al usuario la libre elección del entorno de trabajo; también permite la ampliación y adaptación a nuevos problemas no previstos

cXML fue creado en 1.999 por un consorcio de compañías como Microsoft, VISA, Lucent Technologies, Hewlett Packard, Cisco Systems, Ariba, Philips ó Nestle entre una larga lista de ellas, con el objetivo de definir, implementar y adoptar un protocolo estándar especializado en la consistencia de las comunicaciones de documentos comerciales entre aplicaciones de adquisición, centros de comercio electrónico y proveedores. La impulsora principal de esta iniciativa es Ariba.

La web donde se publican todas las especificaciones de cXML es.

http://www.cXML.org

Actualmente está vigente la versión 1.2.011.

5.1.2. Capacidades principales de cXML

cXML permite la comunicación entre compradores, proveedores, agrupadores de productos e intermediarios mediante un único lenguaje abierto y estándar. Las relaciones de comercio electrónico entre empresas, es decir, el llamado comercio electrónico B2B (*business-to-business*) dependen de un protocolo flexible y de uso generalizado. cXML constituye la clave para proporcionar el máximo acceso a sus productos y servicios, ya que se trata de un eficaz lenguaje bien definido, diseñado específicamente para el comercio electrónico B2B y elegido por proveedores y compradores de grandes volúmenes.

Las transacciones cXML consisten en documentos, que son archivos de texto sencillos con formato y contenido bien definidos. La mayoría de los tipos de documentos cXML son análogos a los documentos impresos empleados tradicionalmente en las empresas.

Los tipos de documentos más usados e importantes en cXML son:

- Catálogos
- Órdenes de compra
- Punchouts

Punchout es un protocolo sencillo para implementar sesiones interactivas de comercio electrónico a través de Internet. Los dos

primeros se describen más en profundidad en los siguientes apartados.

5.1.2.1. Catálogos

Los catálogos son archivos que describen productos o servicios a las organizaciones cliente, especificando el precio y otra información, y son el principal canal de comunicación entre los proveedores y sus consumidores. Los proveedores crean catálogos para que las aplicaciones de adquisición puedan ver los productos y servicios que ofrecen y puedan adquirirlos o contratarlos. Las aplicaciones de adquisición leen los catálogos y los almacenan localmente, de forma que los usuarios puedan añadir productos a los pedidos de compra. Los catálogos pueden describir cualquier tipo de servicio o producto, sin importar cómo esté facturado, medido o enviado. Además, cXML permite soporte multilenguaje.

La siguiente figura describe el proceso de adquisición de catálogos:

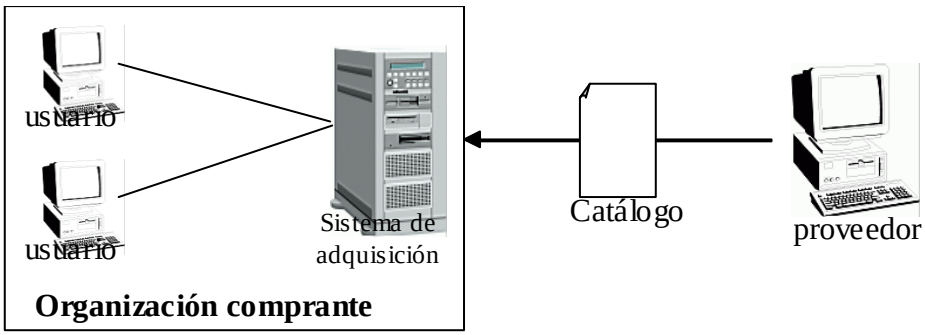


Figura 19: Proceso de envío de catálogos

5.1.2.2. Órdenes de compra

Las órdenes de compra son enviadas por las organizaciones que compran a los proveedores para pedir el cumplimiento de un contrato. cXML es más adecuado para esta tarea que otros estándares (por ejemplo, EDI), ya que es más flexible, barato a la hora de implementar y soporta datos de dispares orígenes.

En la siguiente figura se describe este proceso:

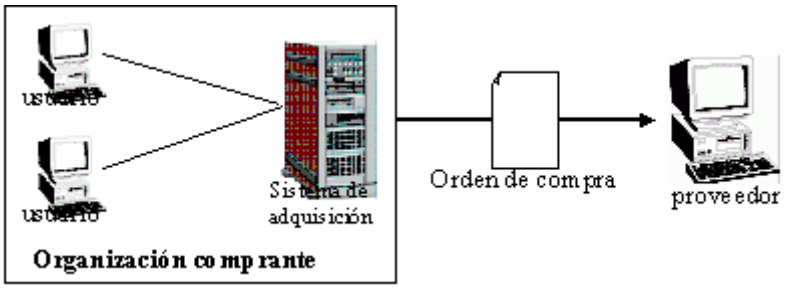


Figura 20: Proceso de envío de órdenes de compra

5.1.3. Aplicaciones que usan cXML

cXML puede ser usado por cualquier aplicación de comercio electrónico. Actualmente es usada por organizaciones comprantes, comunidades de negocios verticales y horizontales, proveeedores y vendedores de aplicaciones. A continuación se detallan algunos casos:

- **Aplicaciones de adquisición**: suelen ser aplicaciones usadas por grandes organizaciones para sus empleados a través de una *intranet*, permitiéndose en ocasiones la creación de subcomunidades especializadas en determinadas áreas de negocio.

- **Concentradores de red de comercio**: son servicios basados en web que conectan organizaciones comprantes y proveedores. Actúan como intermediarios que autentifican y encaminan las peticiones y respuestas de distintas organizaciones.

- **Catálogos Punchout**: son aplicaciones web, escritas en lenguajes como JSP o CGI, que gestionan sesiones interactivas de comercio.

- **Sistemas de recepción de pedidos**: son aplicaciones que aceptan y procesan pedidos enviados por organizaciones comprantes. Son aplicaciones plenamente automatizadas, como bases de datos o gestores de inventarios. Gracias a la sencillez del lenguaje, es fácil realizar la migración a cXML.

5.1.4. Validación XML

Puesto que cXML es un lenguaje XML, un conjunto de definiciones

del tipo de documento (DTD) lo definen a fondo. Estos DTD son archivos de texto que describen la sintaxis y el orden precisos de los elementos de cXML. Los DTD permiten a las aplicaciones validar el cXML que leen o escriben. cXML no define *Schemas* XML puesto que hay escasez de validadores sintácticos que cumplan de forma eficiente las normas de W3C.

No se necesitan aplicaciones cXML para validar documentos cXML, aunque se recomienda. Los DTD para todas las versiones de cXML están disponibles en ubicaciones coherentes en cxml.org:

http://xml.cXML.org/schemas/cXML/<*versión*>/cXML.dtd

Las aplicaciones pueden utilizar estos DTD para validar todos los documentos cXML entrantes y salientes, ya que los mensajes cXML tienen en la cabecera la URL del DTD con el cual debe ser validado.

Para un funcionamiento más robusto, se deberían validar todos los documentos cXML recibidos, aunque en general no es obligatorio, sino recomendado. Sin embargo, todos los documentos cXML deben ser válidos y referirse a los DTD determinados por la versión. En la versión 1.2.009 se especifican 4 DTDs:

- Documento básico cXML: con los elementos básicos de cXML. Es el básico y más importante.
- Confirmaciones y anuncio de envío.
- Facturas
- Definiciones de tipo de catálogos.

Las razones de esta división en 4 DTDs son

- Aumentar la eficiencia de la validación en determinados procesadores XML.
- Potenciar una mayor especialización en el desarrollo de distintos aspectos del comercio electrónico.

Para un funcionamiento más eficiente, las aplicaciones que usen cXML deberían almacenar en caché local los DTDs, debido a que estos DTDs no cambian nunca una vez que son publicados por la organización cXML.org. De esta manera, se adquieren dos grandes ventajas:

- Validación mucho más rápida, puesto que el procesador no tiene ya que acceder a un recurso en la red.

- Menos dependencia en el sitio web cXML.org, ya que si fallase esta web, no podrían validarse los documentos enviados y recibidos, lo que provocaría graves errores en el sistema.

5.2. ASPECTOS BÁSICOS DE CXML

5.2.1. Elementos y atributos

Al estar basado en XML, cXML usa elementos y atributos. Los elementos son usados para describir los conceptos del comercio electrónico, que normalmente son propiedades en los documentos tradicionales usados en la empresa. Los atributos especifican la naturaleza del contenido de un elemento, y añaden también información.

En cXML se adopta la siguiente convención:

- **Elementos**: nombres sin guiones y empezando todas las palabras con mayúsculas.

- Ejemplos: Sender, Credential, IndexItem.

- **Atributos**: nombre sin guiones y empezando la primera palabra por minúscula, las siguientes en mayúscula.

- Ejemplos: version, payloadID, lineNumber.

Existen unos tipos de entidad usados en cXML que son pertenecientes a la W3C, y son usados en muchos elementos y entidades cXML:

- `isoLangCode`: en formato ISO 639.
- `isoCountryCode`: en formato ISO 3166.
- `xmlLangCode`: en ISO 639 o ISO 3166.
- `unitOfMeasure`: formato UN/CEFACT "Unit of Measure Common Codes".
- `URL`: formato definido por HTTP/1.1

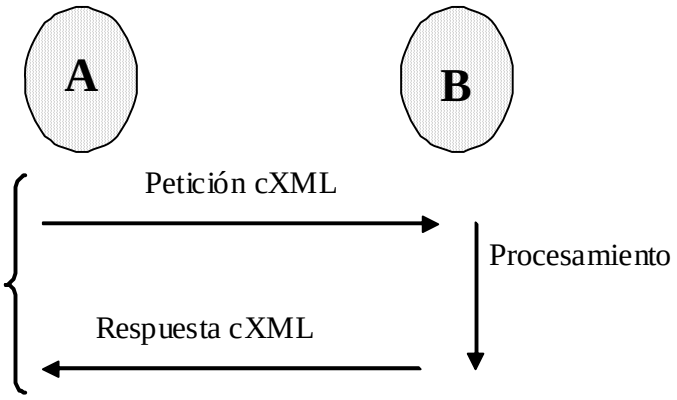
5.2.2. Especificación de protocolo

En principio, cXML sólo especifica el protocolo HTTP para realizar las transacciones de comercio electrónico. En este apartado se explican los modelos de transacción en el caso del uso de HTTP como protocolo de transporte, aunque no se entrará en detalle en los aspectos específicos relacionados con HTTP, con el objetivo de dotar de generalidad a este resumen de los modelos de transacción.

Existen dos modelos de comunicación para las transacciones cXML sobre HTTP: el Petición-respuesta y el Sentido único. Estos dos tipos de modelo permiten realizar de manera sencilla la implementación de intercambios de mensajes cXML, ya que se describen con todo detalle las operaciones requeridas. Ambos modelos son necesarios porque se producen situaciones en que uno solo no sería adecuado.

5.2.2.1. Modelo Petición-Respuesta

Las transacciones de petición-respuesta se pueden realizar mediante una conexión HTTP. La siguiente imagen ilustra los pasos que se deben realizar en una interacción de petición-respuesta entre las partes A y B:



1. A inicia una conexión HTTP/1.x con B en una URL predeterminada que representa la dirección de B.

2. A utiliza una operación POST para enviar el documento cXML mediante la conexión HTTP.

3. A espera que se devuelva una respuesta a través de la conexión HTTP.

4. B tiene un servidor compatible con HTTP/1.x que remite la petición de HTTP al recurso especificado por la URL que se ha especificado en el paso 1. Este recurso puede ser cualquier ubicación conocida por el servidor HTTP de B, como por ejemplo un programa CGI o una página JSP.

5. El recurso de B, identificado en el paso 4, lee el contenido del documento cXML y asigna la petición al gestor adecuado.

6. El gestor de B encargado de esa petición cXML realiza la tarea que la petición especifica y genera un documento cXML como respuesta.

7. B envía la respuesta cXML a A mediante la conexión HTTP establecida en el paso 1.

8. A lee la respuesta cXML y la devuelve al proceso que ha iniciado la petición.

9. A cierra la conexión HTTP establecida en el paso 1. Este proceso se repetirá en futuros ciclos de petición/respuesta.

Figura 21: Modelo de transacción petición-respuesta en HTTP

Para simplificar la tarea en los pasos anteriores, los documentos cXML se dividen en dos partes diferenciadas:

- **Cabecera**: Contiene la información sobre la verificación y la dirección.

- **Datos de la petición o la respuesta**: Contienen una petición

o respuesta específica, así como la información que se debe transferir.

Ambos elementos se incluyen en un elemento de sobre principal. En el siguiente ejemplo se muestra la estructura de un documento de petición cXML:

```
<cXML>
      <Header>
            La información de la cabecera va aquí…
      </Header>
      <Request>
            La información de la petición va aquí…
      </Request>
</cXML>
```

En el siguiente ejemplo se muestra la estructura de un documento de respuesta cXML:

```
      <cXML>
            <Response>
                  La información de respuesta va aquí...
            </Response>
      </cXML>
```

La estructura de la respuesta no utiliza el elemento de cabecera porque la respuesta siempre viaja en la misma conexión HTTP que la petición.

El sobre principal

El elemento de sobre es la raíz de la estructura de los documentos cXML y contiene a los demás elementos. El elemento `cXML` está presente en todas las transacciones cXML. En el siguiente ejemplo se muestra un elemento `cXML` totalmente especificado:

```
      <cXML version="1.0"
            xml:lang="ES"
```

```
        payloadID=1234567.4567.5678@cxml.org.com

        timestamp="2004-04-17T00:47:09+01:00">
```

`cXML` tiene los siguientes atributos:

- **version:** (opcional) Especifica la versión del protocolo cXML. Un analizador XML de validación podría también determinar el atributo de la versión mediante las definiciones del tipo de documento (DTD) a las que hace referencia.

- **xml:lang:** (opcional) Lengua utilizada para todos los textos libres que se han enviado en este documento. El receptor debe responder o mostrar la información en el mismo lugar o en uno similar.

- **payloadID:** Número único, en relación con el espacio y el tiempo, que se utiliza en las operaciones de conexión para identificar documentos que pueden haberse perdido o tener problemas. Este valor no debería cambiar en caso de que se vuelva a intentar.

- **Timestamp:** Fecha y hora en que se envió el mensaje. Este valor no debería cambiar en caso de que se vuelva a intentar. Este atributo y el resto de las fechas y horas en cXML deben seguir el formato que dicta el subconjunto restringido de ISO 8601 descrito en la nota emitida por el W3C con título "Date and Time Formats" (formatos de fecha y hora) disponible en www.w3.org/TR/NOTE-datetime-970915.html.

- Los registros horarios requieren como mínimo una fecha completa más hora, minutos y segundos. Las fracciones de segundo son opcionales. Este protocolo necesita que las horas estén expresadas en hora local con un desfase de zona horaria respecto de la UTC también denominada como GMT. El indicador de zona horaria "Z" no está permitido. Por ejemplo, `17-04-2004T01:09:00-08:00` corresponde al 17 de abril de 2004, 1:09 a.m., hora oficial del Pacífico.

La cabecera de la petición cXML

El elemento `Header` contiene información sobre la dirección y la verificación. Este elemento es el mismo independientemente de la petición o respuesta específicas que se contengan en el cuerpo del mensaje cXML. Las aplicaciones necesitan la identidad del solicitante, pero no es necesario comprobar que la información sobre la identidad proporcionada es correcta.

En el siguiente ejemplo se muestra el elemento `Header`:

```
<Header>
      <From>
            <Credential domain="Dominio0">
                  <Identity>usuario1@acme.com</Identity>
            </Credential>
      </From>
      <To>
            <Credential domain="Dominio1">
                  <Identity>012345678</Identity>
            </Credential>
      </To>
      <Sender>
            <Credential domain="Dominio2">
                  <Identity>servidor1@comerciop2p.com</Identity>
                  <SharedSecret>abracadabra</SharedSecret>
            </Credential>
            <UserAgent>ComercioP2P 1.0</UserAgent>
      </Sender>
</Header>
```

Los elementos `From` y `To` son sinónimos de los elementos `From` y `To` de los mensajes de correo SMTP, que son la fuente lógica y el destino de los mensajes. `Sender` es la parte que inicia la conexión HTTP y envía el documento cXML.

Sender contiene el elemento Credential, mediante el cual la parte receptora puede verificar la parte emisora. Este elemento permite realizar verificaciones de gran envergadura sin necesidad de una potente infraestructura de certificado digital de clave pública. Sólo se debe proporcionar un nombre de usuario y una clave mediante el elemento SharedSecret en la parte receptora para permitir a la enviante el envío de peticiones. El elemento Credential tiene el atributo domain que especifica el tipo de credencial, permitiendo que un mismo documento cXML contenga varios tipos de credenciales. Además, se permite una autentificación más robusta mediante un elemento CredentialMac que indica que se ha implementado un método de autenticación MAC (Código de Autentificación de Mensaje), basado en una entidad tercera de confianza que actúa como intermediaria.

El cuerpo de la petición cXML

Los clientes envían peticiones para solicitar operaciones. Sólo se permite un elemento `Request` por cada elemento de envoltura cXML, lo que repercute en la simplicidad de las implementaciones de los servidores, ya que no hay entonces que realizar demultiplexión a la hora de analizar y procesar los documentos cXML. El elemento `Request` puede contener virtualmente cualquier tipo de datos XML.

Algunos ejemplos de elementos `Request` tipo son:

- OrderRequest
- ProfileRequest
- PunchOutSetupRequest
- SubscriptionContentRequest
- SupplierListRequest

Cada tipo de elemento tiene sus propios elementos de contenido.

El cuerpo de la respuesta cXML

Los servidores envían respuestas para informar a los clientes de los resultados de las operaciones. Como puede ocurrir que los resultados

de algunas peticiones no tengan datos, el elemento `Response` puede contener de manera opcional sólo un elemento `Status`. El elemento `Response` también puede contener datos de cualquier nivel de la aplicación.

El elemento `Status` tiene la siguiente forma:

```
<Status code="406" text="Not Acceptable">No se pudo validar!</Status>
```

- **code:** Código del estado de la petición. Sigue el modelo de códigos de estados HTTP. Por ejemplo, `200` representa una petición correcta.

- **text:** Texto del mensaje de estado. Este texto facilita al usuario la lectura de los registros y está compuesto de cadenas canónicas en inglés.

Los atributos del elemento `Status` indican lo que le ha sucedido a la petición. Los datos del elemento `Status` pueden ser cualquier información que necesite el solicitante. Para el código de estado `200/OK`, puede que no haya datos. Sin embargo, para el código de estado `500/Internal Server Error`, se recomienda que se especifique el error de análisis XML o el error de aplicación. Este error permite realizar mejor la depuración unilateral y las pruebas de interoperabilidad. Los servidores no deberían incluir elementos de respuesta adicionales a menos que el código de estado se encuentre en el rango `200` (Por ejemplo, `200/OK`).

5.2.2.2. Modelo sentido único (asíncrono)

A diferencia de las transacciones de petición-respuesta, los mensajes de sentido único no se limitan al transporte HTTP. Estos mensajes se emplean cuando no es adecuado utilizar un canal HTTP (una operación del tipo petición-respuesta sincrónico). La siguiente figura muestra un ejemplo de cómo A y B se pueden comunicar con mensajes en lugar de hacerlo a través de una transacción de petición-respuesta.

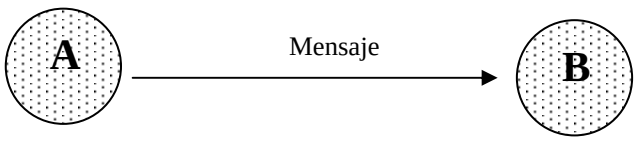


Figura 22: Modelo de transacción asíncrono

En el modelo sentido único, A y B no tienen un ciclo explícito de petición-respuesta. Pueden llegar mensajes procedentes de terceros y se pueden producir conversaciones con otros.

Los mensajes de sentido único tienen una estructura similar a la del modelo petición-respuesta, pero tienen como cuerpo sólo un elemento `Message`:

```
<cXML>
      <Header>
            La información de la cabecera va aquí…
      </Header>
      <Message>
            La información del mensaje va aquí…
      </Message>
</cXML>
```

5.3. TRANSACCIONES CXML

5.3.1. Transacción de Perfil

Las implementaciones del servidor cXML desde la versión 1.1 en adelante deben admitir los documentos `ProfileRequest` y `ProfileResponse`. Esta transacción utiliza para recuperar las capacidades del servidor, incluyendo la versión cXML admitida, las transacciones y las opciones de éstas. Es la única transacción cXML que debe ser implementada obligatoriamente en todo servidor cXML, y su propósito es proporcionar compatibilidad entre aplicaciones, de forma que el cliente tenga conocimiento de las capacidades soportadas por el servidor cXML con el que está conversando.

La respuesta debe contener todas las peticiones admitidas por el servidor, no necesariamente todas las que admite la empresa. Los proveedores que pueden recibir documentos `OrderRequest` o iniciar transacciones de petición-respuesta describen su soporte respecto a `OrderRequest` en la transacción de perfil.

La transacción de perfil se puede utilizar para realizar un *ping* al servidor dentro del protocolo cXML.

5.3.1.1. Elemento ProfileRequest

Este elemento no tiene contenido. Tan sólo se dirige al servidor cXML adecuado utilizando información contenido en el elemento `Header`. El servidor responde con una sola `ProfileResponse`. Las únicas partes dinámicas de esta respuesta son los atributos `payloadId` y `timestamp` del propio elemento `cXML`.

Un ejemplo de petición de este tipo sería el siguiente:

```
<Request>
      <ProfileRequest />
</Request>
```

5.3.1.2. Elemento ProfileResponse

Este elemento contiene una lista de transacciones admitidas, sus ubicaciones y las opciones admitidas. El siguiente ejemplo es una posible `ProfileResponse`:

```
<ProfileResponse effectiveDate="2001-03-03T12:13:14-05:00">
      <Transaction requestName="PunchOutSetupRequest">
            <URL>http://www.workchairs.com/comprar.jsp</URL>
      </Transaction>
</ProfileResponse>
```

El atributo `effectiveDate` indica la fecha y hora (en formato ISO 8601) desde la que esta transacción está disponible. El elemento `Transaction` da la descripción de la transacción soportada, en particular mediante el atributo `requestName`, que proporciona la información sobre el nombre de la transacción soportada, y el elemento `URL`, que indica la URL donde está disponible la transacción.

Existen además otros elementos que proporcionan información adicional sobre las transacciones en los que no se entrará en más detalles.

5.3.2. Órdenes de compra

Los documentos de pedidos cXML son `OrderRequest` y una

respuesta genérica cXML. `OrderRequest` es similar a un pedido de compra. La respuesta es una confirmación de que el proveedor ha recibido el pedido de compra. No es necesario ejecutar el pedido de compra, sino la confirmación de que se ha recibido correctamente.

5.3.2.1. Elemento OrderRequest

En el siguiente ejemplo se muestra la estructura del elemento `OrderRequest`:

```
<OrderRequest>
      <OrderRequestHeader … >
            Cabecera de la petición
      </OrderRequestHeader>
      <ItemOut … >
            Información de un producto pedido
      </ItemOut>
      <ItemOut … >
            Información de un producto pedido
      </ItemOut>
</OrderRequest>
```

El elemento `OrderRequest` tiene, entre otros, dos atributos importantes: `orderID`, que identifica una petición de compra, y `orderDate`, que da la fecha de la orden (en formato ISO 8601, como todas las fechas en cXML).

Es muy similar a cualquier pedido de compra comercial usado en documentos sobre papel. Contiene dos elementos principales, que son:

- **OrderRequestHeader**: una cabecera con información genérica del pedido. Contiene los siguientes elementos:
  - `Total`: el total del pedido, excluyendo impuestos y gastos de envío.
  - `ShipTo` (?): la dirección de la entidad a la que se enviará el pedido.

- `BillTo`: la dirección de la entidad a la que se facturará el pedido.

- `Shipping` (?): el método de envío para el pedido.

- `Tax` (?): información sobre los impuestos, si el sistema admite computación de los mismos.

- `Payment` (?): información sobre el método de pago. cXML sólo soporta elementos de pago electrónico, aunque se prevé que en el futuro se añadirán otros.

- `Contact` (*): información de contacto para seguir y rastrear una orden de compra.

- `Comments` (?): información arbitraria legible por personas, es decir, no está pensado para procesamiento por computador.

- `Followup`: información para seguimiento.

- `Extrinsic` (*): información relacionada con el pedido que debe ser legible para una máquina que procese órdenes cXML.

- **ItemOut** (+): información específica de un producto adquirido. Tiene un atributo `quantity` que indica el número deseado de artículos de este ítem. Los elementos que incorpora son:

  - `ItemID`: el identificador interno del producto.

  - `ItemDetail` (?): información que describe al producto, contenida en el catálogo.

  - `SupplierID` (?): el identificador interno del proveedor.

  - `ShipTo` (?): la dirección de la entidad a la que se enviará el pedido.

- `Shipping` (?): el método de envío para el pedido.
- `Tax` (?): información sobre los impuestos, si el sistema admite computación de los mismos.
- `Distribution` (*): información sobre la división del precio en distintas partes.
- `Contact` (*): información de contacto para seguir y rastrear una orden de compra.
- `Comments` (?): información arbitraria legible por personas, es decir, no está pensado para procesamiento por computador.

La leyenda para los signos es:

- "+" : el elemento puede ocurrir una o más veces.
- "?" : el elemento puede ocurrir una o ninguna vez.
- "*" : el elemento puede ocurrir ninguna o más veces.

Esto implica que una orden de compra cXML es muy configurable, de forma que una organización puede incluir sólo un subconjunto de los elementos posibles, adaptándolo a la información que desean realmente transmitir. Cada elemento tiene sus propios atributos y subelementos cXML, que no serán tratados aquí.

5.3.2.2. Respuesta a una OrderRequest

Ésta es la parte de respuesta de la transacción de petición-respuesta síncrona. En el siguiente ejemplo se muestra la respuesta de un documento `OrderRequest`:

```
<cXML version="1.2" payloadID="9949494" xml:lang="es"
timestamp="1999-03-12T18:39:09-08:00">
      <Response>
            <Status code="200" text="OK"/>
```

```
        </Response>

    </cXML>
```

La respuesta es sencilla y con propósitos de confirmación. No existen elementos reales denominados "`OrderResponse`", porque el único dato que se debe devolver al solicitante es la parte `Status` de la respuesta. La respuesta comunica al solicitante que la parte remota de la conexión ha recibido y aceptado el pedido cXML. Sin embargo, no comunica la confirmación en el nivel de pedido, es decir, que el pedido realmente haya sido aceptado porque se dan las condiciones para que sea posible (que haya existencias, que no haya problemas de transporte, etc).

5.3.3. Catálogos

5.3.3.1. Elementos básicos

Las definiciones del catálogo cXML constan de dos elementos principales: `Supplier`, `Index` y `Contract`. Todos estos elementos describen los datos concebidos para uso permanente o temporal en el sistema de adquisición de un comprador o un concentrador.

- **`Supplier`**: Contiene datos básicos acerca del proveedor, como la información sobre la realización de pedidos, la dirección y el contacto.

- **`Index`**: Describe los datos acerca del inventario de productos del proveedor, como por ejemplo descripciones, números de pieza y códigos de clasificación. Utiliza varios subelementos para describir los artículos de línea en los inventarios del proveedor.

El elemento `Supplier` contiene 2 elementos fundamentales:

- `Name`: el nombre del proveedor.

- `SupplierID`: contiene un identificador para el proveedor, con su respectivo dominio de identidad.

- `SupplierLocation`: contiene información relativa a la dirección física del proveedor, así como a la dirección

virtual, con datos como las direcciones de los sitios de compra y los métodos admitidos.

El elemento `Index` es el que gestiona la información de un catálogo. Se asocia generalmente a un solo proveedor. Contiene uno o más elementos `IndexItem`, que a su vez contienen uno o varios elementos que añaden o eliminan productos del catálogo que está en el caché de la organización que compra. Estos elementos pueden ser:

- `IndexItemAdd`: inserta o actualiza un producto. Contiene elementos con la identificación del producto, la descripción, e información para el catálogo.

- `IndexItemDelete`: borra un producto del catálogo dado su identificador de producto.

- `IndexItemPunchout`: especifica un producto para *punchout*.

5.3.3.2. Gestión de suscripciones

Los intermediarios pueden gestionar la información de catálogos y proveedores usadas por las aplicaciones de adquisición. Estas transacciones son iniciadas por los sistemas de adquisición. Existen varios ciclos de petición/respuesta, entre los que destacan:

- **Lista de Proveedores**: Se compone de una petición con el elemento `SupplierListRequest` y una respuesta con el elemento `SupplierListResponse`. Se realiza un intercambio de información de la lista de proveedores con los que el comprador ha establecido relaciones comerciales.

- **Datos de Proveedores**: Se compone de una petición con el elemento `SupplierDataRequest` y una respuesta con el elemento `SupplierDataResponse`. Se realiza un intercambio de información del proveedor especificado por un identificador. La respuesta contiene el elemento `Supplier`, con datos como el nombre del proveedor, dirección, formas de ordenar pedidos, etc.

- **Lista de Catálogos**: Se compone de una petición con el elemento `SubscriptionListRequest` y una respuesta con el elemento `SubscriptionListResponse`. Se pide la lista de

catálogos a los que el comprador está suscrito.

- **Datos de Catálogo**: Se compone de una petición con el elemento `SubscriptionContentRequest` y una respuesta con el elemento `SubscriptionContentResponse`. Se pide el contenido de un determinado catálogo de un proveedor también especificado. Se devuelve una respuesta con el contenido del catálogo en sí. El catálogo puede estar expresado en dos formatos:

  - CIF (Catalog Interchange Format): estará incluido en un elemento `CIFContent` y codificado en base64.

  - cXML: se incluye directamente el elemento `Index`.

5.3.4. Otras transacciones

Existen otras transacciones disponibles en cXML que no han sido comentadas con más detalle ya que no se han usado en el desarrollo de este proyecto. Algunas de ellas son muy potentes y convierten a cXML en una opción muy recomendable para la implementación de aplicaciones de comercio electrónico. Se pueden citar:

- **Transacción Punchout:** Es una transacción muy interesante y usada. Muchas empresas usan cXML para implementar el soporte para sitios `punchout`. Permite iniciar y mantener sesiones de comercio electrónico con páginas web del proveedor. De esta manera se envía un pequeño índice de los productos, en vez de enviar un catálogo completo, usando por tanto catálogos interactivos (residentes en el proveedor) en vez de estáticos. Las aplicaciones de adquisición muestran un botón en lugar de detalles sobre productos o precios. Cuando los usuarios hacen clic en este botón, el navegador Web empleado muestra páginas de su sitio Web local. Según cómo implemente las páginas, los usuarios podrán navegar por las opciones de productos, especificar configuraciones y seleccionar métodos de entrega. Cuando los usuarios hayan terminado de seleccionar artículos, pueden hacer clic en un botón que devuelve la información del pedido a la aplicación de adquisición. Los productos configurados totalmente y sus precios aparecen en las solicitudes de compra de los usuarios.

- **Transacción MasterAgreement:** realiza una negociación para establecer un acuerdo de negocio con un proveedor. Representa un mecanismo para la gestión de compromisos con el proveedor, y permite la negociación de descuentos basándose en futuras adquisiciones.

- **Transacción de Cambios de estado:** Regula el intercambio de información adicional que se puede producir una vez finalizada la transacción de `OrderRequest`. Es posible que los servidores intermediarios tengan que proporcionar información extra al sistema comprador como confirmaciones de estado del pedido, del envío, etc.

- **Transacción de Facturas:** se utiliza para el envío de documentos de facturas a las entidades participantes en el proceso de negocio. Estos documentos pueden ser facturas estándar, recibos, documentos de crédito o débito, etc.

- **Transacción de Recuperación de mensajes:** permiten a los sistemas la recuperación de mensajes que permanecen en la cola de entrada, en el caso de que los compradores no dispongan de puntos de entrada HTTP para mensajes cXML.

Como puede observarse el lenguaje cXML es muy potente y permite transacciones que no se van a implementar en este proyecto. En la aplicación desarrollada se hará uso solamente de la transacción de compra y de la transacción de perfil, esta última es la única que el éstandar indica que hay que implementar obligatoriamente en todo servidor cXML.

6. JXTA

Para la implementación de la tecnología peer-to-peer en este proyecto se ha escogido JXTA, ya que actualmente no existe ningún otro marco de trabajo con características similares. JXTA define un conjunto de protocolos de carácter general, y la comunidad JXTA proporciona las implementaciones de estos protocolos en distintos lenguajes.

A continuación se detallan los aspectos básicos de esta tecnología, su arquitectura y los diferentes protocolos.

6.1. INTRODUCCIÓN

JXTA es un conjunto abierto y generalizado de protocolos Peer-to-Peer (P2P, par-a-par) que permiten a cualquier dispositivo conectado en la red (desde un teléfono móvil a un PDA, desde un PC a un servidor) comunicarse y colaborar como pares. Los protocolos de JXTA son independientes de cualquier lenguaje de programación, y existen múltiples aplicaciones para los diferentes entornos. Aquí se discute específicamente el Proyecto JXTA sobre Java™ 2 Platform, Standard Edition software (J2SE™).

6.1.1. ¿Por qué el Proyecto JXTA?

Como la Web continúa creciendo en contenido y en número de dispositivos conectados, las redes par-a-par están aumentando en popularidad. El Software actual basado en las tecnologías P2P incluye compartición de archivos, computación distribuida, y los servicios de mensajería instantánea. Mientras cada una de estas aplicaciones realiza tareas diferentes, todos ellos comparten muchas propiedades, como el descubrimiento de pares, investigación, y transferencia de archivos o de datos. Actualmente, el desarrollo de la aplicación es ineficaz, con desarrolladores que resuelven los mismos problemas y duplicando implementaciones similares de la infraestructura. Y, la mayoría de las aplicaciones son específicas a una sola plataforma y son incapaces de comunicarse y compartir los datos con otras aplicaciones.

Un objetivo primario del Proyecto JXTA es proporcionar una plataforma con las funciones básicas necesarias para una red P2P. Además, la tecnología JXTA busca superar las limitaciones potenciales en muchos de los sistemas de P2P existentes:

- Interoperabilidad. La tecnología de JXTA esta diseñada para permitir a los pares proporcionar varios servicios P2P para localizar a los otros pares y comunicarse entre sí.

- Independencia de la plataforma: La tecnología JXTA esta diseñada para ser independiente de los lenguajes de programación, protocolos de transporte, y plataformas de ejecución.

- Ubicuidad: La tecnología de JXTA esta diseñada para ser accesible por cualquier dispositivo digital, no sólo PCs o una plataforma de ejecución específica.

Una característica común de los pares en una red P2P es que estos a menudo están en el borde de la red regular. Debido a que están sujetos a una conectividad imprevisible con direcciones de red potencialmente variables, están fuera del alcance estándar del DNS. JXTA proporciona un sistema de direccionamiento único, independiente de los servidores de nombre tradicionales, para los pares que están en el borde de la red. A través del uso de Identificaciones JXTA, un par puede vagar por las redes, cambiando direcciones de transporte y de red, incluso ser temporalmente desconectado, y ser todavía direccionable por otros pares.

6.1.2. ¿Qué es el Proyecto JXTA?

El proyecto JXTA es una plataforma de programación de red abierta diseñada para la programación peer-to-peer (P2P). Su meta es desarrollar bloques constructivos básicos y servicios para permitir aplicaciones innovadoras para grupos pares. El término "JXTA" es una abreviación de yuxtaponer. Está reconocido que P2P es yuxtapuesta a la programación cliente-servidor o a la programación basada en la Web, las cuales son el modelo de la programación distribuida de hoy.

JXTA proporciona un conjunto común de protocolos abiertos y una fuente de implementaciones de referencia abierta para desarrollar aplicaciones peer-to-peer. Los protocolos JXTA estandarizan la manera en que los pares:

- Se descubren unos a otros.

- Se Auto-organizan en grupos de pares.

- Anuncian y descubren servicios de red.
- Se comunican entre sí.
- Se controlan mutuamente.

Los protocolos de JXTA están diseñados para ser independientes de los lenguajes de programación, e independientes de los protocolos de transporte. Los protocolos pueden implementarse en lenguaje de programación Java, C/C++, Perl, y numerosos otros lenguajes. Estos pueden implementarse sobre TCP/IP, HTTP, Bluetooth, HomePNA, u otros protocolos de transporte.

6.1.3. ¿Qué puede hacerse con la Tecnología JXTA?

Los protocolos JXTA permiten a los desarrolladores construir y ejecutar servicios y aplicaciones P2P interoperables. Debido a que los protocolos son independientes del lenguaje de programación y protocolos de transporte, los dispositivos heterogéneos con pilas de software completamente diferentes pueden interoperar unos con otros. Usando JXTA, los desarrolladores pueden escribir aplicaciones de red interoperables que pueden:

- Encontrar otros pares en la red atravesando firewalls con el descubrimiento dinámico.
- Compartir fácilmente los documentos con cualquier otro por la red.
- Encontrar al momento contenidos en sitios de la red.
- Crear un grupo de pares que proporcionan un servicio.
- Monitorizar actividades de pares remotamente.
- Comunicarse de forma segura con otros pares en la red.

6.1.4. Historia de JXTA

JXTA nació en el seno de Sun Microsystems como un proyecto de

investigación de la tecnología P2P. Desde el principio, se formó una comunidad de desarrollo interesada en la tecnología. En abril de 2001, se abre la web del proyecto JXTA, incluyendo las especificaciones de los protocolos y animando a desarrolladores de todo el mundo a unirse en los esfuerzos por implementar nuevos servicios y plataformas con la nueva tecnología. La comunidad JXTA es ahora la que tiene la responsabilidad de desarrollar el proyecto.

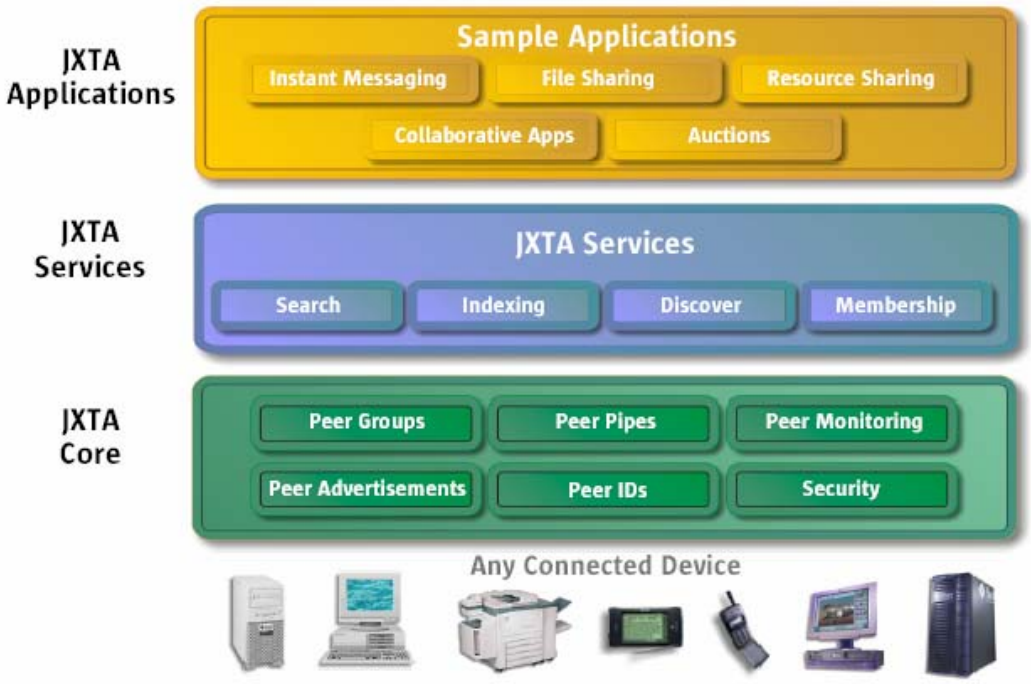
JXTA está fuertemente apoyado por Sun, y se han unido al apoyo numerosas universidades y empresas privadas de software.

Actualmente, multitud de proyectos están siendo llevados a cabo dentro del ámbito de la comunidad JXTA, desde las implementaciones en lenguajes como Python hasta motores de búsqueda en internet.

6.2. LA ARQUITECTURA DE JXTA

6.2.1. Visión General

La arquitectura software del Proyecto JXTA está dividida en tres capas, mostradas en la Figura:



- **Capa de plataforma (JXTA Core)**. La capa de plataforma, también conocido como JXTA core, encapsula las primitivas mínimas y esenciales comunes en una red P2P. Incluye los

bloques constructivos que habilitan los mecanismos importantes de las aplicaciones P2P, incluyendo descubrimiento, transporte (incluso manejo de cortafuegos), creación de pares y de grupos de pares, y las primitivas asociadas a la seguridad.

- **Capa de servicios**. La capa de servicios incluye servicios de red que pueden no ser absolutamente necesarios para que funcione una red P2P, pero es común o deseable en el ámbito P2P. Ejemplos de servicios de red incluyen búsqueda e indexación, directorio, sistemas de almacenamiento, compartición de archivos, sistemas de archivo distribuidos, agregación y alquiler de recursos, traducción de protocolos, autentificación, y los servicios PKI (Infraestructura de clave pública).

- **Capa de aplicación**. La Capa de aplicación incluye la implementación de aplicaciones integradas, como la mensajería instantánea P2P, la compartición de archivos y recursos, la gestión y entrega de contenidos lucrativos, los sistemas de E-mail P2P, los sistemas de subastas distribuidas, y muchos otros.

El límite entre los servicios y las aplicaciones no es rígido. Una aplicación para un cliente puede verse como un servicio por otro cliente. El sistema entero está diseñado para ser modular, permitiendo a diseñadores escoger entre una colección de servicios y aplicaciones que satisfagan sus necesidades.

6.2.2. Componentes de JXTA

La red JXTA consiste en una serie de pares interconectados, o peers. Los pares pueden auto-organizarse en grupos de pares que proporcionan un conjunto común de servicios. Ejemplos de servicios que podrían proporcionarse por un grupo de pares incluyen compartición de archivos o aplicaciones de Chat.

Los pares de JXTA anuncian sus servicios con documentos XML llamados anuncios. Los anuncios permiten a otros pares en la red aprender a conectarse, y actuar recíprocamente, con los servicios de un par.

Los pares JXTA usan pipes (tuberías) para enviarse mensajes entre

sí. Los pipes son un mecanismo de transferencia de mensajes asíncrono y unidireccional usado para la comunicación. Los mensajes son simples documentos XML cuya cabecera contiene la ruta, resumen (digest), e información de la credencial. Los pipes son asociados a puntos finales específicos (endpoints), como un puerto TCP y la dirección IP asociada.

Estos conceptos se describen en detalle en los capítulos siguientes.

6.2.2.1. Aspectos importantes de la arquitectura de JXTA

Tres aspectos esenciales de la arquitectura JXTA lo distinguen de otros modelos de red distribuidos:

- El uso de documentos XML (los anuncios) para describir los recursos de la red.

- La abstracción de pipes a los pares, y pares a puntos finales sin delegar sobre una autoridad central del nombrado/direccionado como DNS.

- Un esquema de direccionamiento único de pares (ID's de pares).

6.3. CONCEPTOS JXTA

Este capítulo define la terminología clave de JXTA y describe los componentes primarios de la plataforma de JXTA.

6.3.1. Peers

Un peer (par) es cualquier dispositivo de red que lleva a cabo uno o más de los protocolos de JXTA. Los pares pueden englobar sensores, teléfonos, y PDA's, así como PC's, servidores, y supercomputadoras. Cada par opera de forma independiente y asíncrona con respecto a los demás pares, y es unívocamente identificado por un identificador de par.

Los pares publican una o más interfaces de red para usar con los protocolos JXTA. Cada interfaz publicada es anunciada como un punto final del par que identifica de forma única la interfaz de red. Los puntos finales del par son usados por estos para establecer conexiones directas

punto a punto entre dos pares.

No se exige a los pares que tengan conexiones de red directas punto a punto entre ellos. Pueden usarse pares intermediarios para dirigir los mensajes a pares que están separados debido a las conexiones físicas de la red o a la configuración de la red (Por ejemplo, NATS, firewalls, proxies).

Los pares están típicamente configurados para descubrir espontáneamente en la red a los otros pares para formar asociaciones temporales o persistentes llamadas grupos de pares.

6.3.2. Grupos de pares

Un grupo de pares es un conjunto de pares que han acodado un conjunto común de servicios. Los pares se auto-organizan en grupos de pares, cada uno es identificado por un único identificador de grupo. Cada grupo de pares puede establecer su propia política asociación desde abierto (cualquiera se puede unir) a altamente seguro y protegido (se exigen credenciales suficientes para unirse).

Los pares pueden pertenecer simultáneamente a más de un grupo. Por defecto, el primer grupo que es instanciado es el Net Peer Group (Grupo de red). Todos los pares pertenecen al Net Peer Group. Los pares pueden elegir unirse a grupos de pares adicionales.

Los protocolos de JXTA describen cómo los pares pueden publicar, descubrir, unirse, y monitorizar grupos de pares; estos no fijan cuando o por qué se crean los grupos de pares.

Hay diferentes motivaciones para crear grupos de pares:

- **Para crear un entorno seguro**. Los grupos crean un dominio local de control en el que puede obligarse a una política de seguridad específica. La política de seguridad puede ser tan simple como intercambiar username/password, o tan sofisticado como la encriptación de clave pública. Las fronteras de los grupos permiten a los pares asociados acceder y publicar contenidos protegidos. Los grupos de pares forman regiones lógicas cuyas fronteras limitan el acceso a los recursos del grupo.

- **Para crear un ámbito de alcance**. Los grupos permiten el establecimiento de un dominio local de especialización. Por ejemplo, los pares pueden agruparse para llevar a cabo una red de compartición de archivos o una red de compartición de procesamiento. Los grupos de pares sirven para subdividir la red en regiones abstractas proporcionando un mecanismo implícito de alcance. Los límites del grupo definen el alcance de la búsqueda al buscar el contenido de un grupo.

- **Para crear un entorno supervisado**. Los grupos permiten a los pares supervisar un conjunto de pares para cualquier propósito especial (por ejemplo, introspección del tráfico, o responsabilidad).

Los grupos también forman una relación de padre-hijo jerárquica en la que cada grupo tiene un sólo padre. Se propagan las solicitudes de búsqueda dentro del grupo. El anuncio del grupo se publica en el grupo padre además de publicarse en el propio grupo.

Un grupo de pares proporciona que un conjunto de servicios llamados servicios de grupo. JXTA define un conjunto básico de servicios de grupo.

Pueden desarrollarse servicios adicionales para proporcionar servicios específicos. Para que dos pares interactúen por medio de un servicio, los dos deben ser parte del mismo grupo del par.

Los servicios principales de grupo son:

- **Discovery Service** (Servicio de descubrimiento). El servicio de descubrimiento se usa por los miembros del grupo para buscar los recursos del grupo, como pares, grupos de pares, pipes y servicios.

- **Membership Service** (Servicio de afiliación a un grupo). El servicio de afiliación se usa por los socios actuales para rechazar o aceptar una nueva solicitud de afiliación. Los pares que deseen unirse a un grupo de pares deben localizar a un socio actual primero, y entonces pedir unirse. La solicitud de afiliación o es rechazada o es aceptada por el conjunto colectivo de socios actuales. El servicio de afiliación puede forzar una votación de los pares o puede

elegir a un grupo representante designado para aceptar o rechazar las nuevas solicitudes de afiliación.

- **Access Service** (Servicio de acceso). El servicio de acceso se usa para validar solicitudes hechas por un par a otro. El par que recibe la solicitud proporciona las credenciales del par que hace la solicitud e información sobre la solicitud que hecha para determinar si el acceso se permite. No todas las acciones dentro del grupo necesitan ser verificadas con el servicio de acceso; sólo aquellas acciones que se limitan a algunos pares necesitan ser verificadas.

- **Pipe Service** (Servicio de tuberías**)**. El servicio de pipes se usa para crear y gestionar las conexiones de los pipes entre los miembros del grupo.

- **Resolver Service** (Servicio de Resolución**)**. El servicio de resolución se usa para enviar las solicitudes de consulta genéricas a otros pares. Los pares pueden definir y pueden intercambiar las consultas para encontrar cualquier información que pueda necesitarse (por ejemplo, el estado de un servicio o el estado de un punto final de pipe).

- **Monitoring Service** (Servicio de Monitorización). El servicio de monitorización se usa para permitirle a un par supervisar a otros miembros del mismo grupo.

No todos los servicios anteriores deben implementarse por cada grupo de pares. Un grupo de pares es libre de implementar sólo los servicios que encuentre útiles, y deja en manos del net peer group inicial el proporcionar implementaciones genéricas de los servicios no críticos del núcleo.

6.3.3. Servicios de red

Los pares cooperan y se comunican para publicar, descubrir, e invocar los servicios de red. Pueden publicar múltiples servicios. Los pares descubren los servicios de la red por medio del Protocolo de Descubrimiento.

Los protocolos de JXTA reconocen dos niveles de servicios de red:

- **Servicios de par.** Un servicio de par sólo es accesible en el par que está publicando ese servicio. Si ese par fallara, el servicio también fallaría. Pueden ejecutarse múltiples instancias del servicio en pares diferentes, pero cada instancia publica su propio anuncio.

- **Servicios de grupo.** Un servicio de grupo está compuesto de una colección de instancias (potencialmente cooperando entre sí) del servicio ejecutándose en múltiples miembros del grupo. Si algún par falla, el servicio de grupo no es afectado (suponiendo que el servicio todavía está disponible en otro miembro del grupo). Los servicios de grupo se publican como parte del anuncio del grupo.

Los servicios pueden o bien preinstalarse en un par o bien cargarse de la red. Para ejecutar realmente un servicio, un par puede tener que localizar una implementación conveniente para el entorno de ejecución del par. El proceso de encontrar, descargar, e instalar un servicio de la red es similar a realizar una búsqueda en Internet para una página Web, primero recuperar la página, y después instalar el plug-in requerido.

6.3.4. Módulos

Los módulos de JXTA son una abstracción que representa a cualquier pieza de "código" usada para llevar cabo un comportamiento en el mundo de JXTA. Los servicios de red son el ejemplo más común de comportamiento que puede ser llevado a cabo en un par. La abstracción de módulo no especifica lo que es este "código": puede ser una clase de Java, un archivo jar de Java, un DLL de la biblioteca dinámica, un conjunto de mensajes XML, o un script. La implementación del comportamiento del módulo se deja a los que implementen el módulo. Por ejemplo, pueden usarse los módulos para representar implementaciones diferentes de un servicio de red en plataformas diferentes, como la plataforma de Java, Microsoft Windows, o Solaris.

Los módulos proporcionan una abstracción genérica para permitir a un par instanciar un nuevo comportamiento. Cuando los pares exploran o se meten en un nuevo grupo, pueden encontrar nuevos comportamientos que pueden querer instanciar. Por ejemplo, al meterse en un grupo, un par puede tener que aprender un nuevo servicio de búsqueda que sólo se usa en este grupo. Para meterse en este grupo, el par debe instanciar este nuevo servicio de búsqueda. La estructura de módulos permite la representación y anuncio de

comportamientos de plataforma-independiente, y permite a los pares describir e instanciar cualquier tipo de implementación de un comportamiento. Por ejemplo, un par tiene la capacidad de instanciar una implementación del comportamiento en Java o en C.

La capacidad para describir y publicar el comportamiento de plataforma-independiente es esencial para soportar grupos compuestos de pares heterogéneos. Los anuncios de módulo posibilitan a los pares de JXTA describir un comportamiento en un modo de plataforma-independiente. La plataforma de JXTA usa los anuncios de módulo para auto-describirse.

La abstracción en módulos incluye module class, module specification, y module implementation:

6.3.4.1. Module Class:

La clase módulo se usa principalmente para anunciar la existencia de un comportamiento. La definición de la clase representa un comportamiento y un enlace esperado para soportar el módulo. Cada clase módulo se identifica por un identificador único, el ModuleClassID.

6.3.4.2. Module Specification:

La especificación de módulo se usa principalmente para acceder a un módulo. Contiene toda la información necesaria para acceder o invocar el módulo. Por ejemplo, en el caso de un servicio, la especificación de módulo puede contener un anuncio del pipe para usarlo para comunicarse con el servicio.

Una especificación de módulo es un acercamiento para proporcionar la funcionalidad que una clase módulo implica. Puede haber múltiples especificaciones de módulo para una clase módulo dada. Cada especificación de módulo se identifica por un identificador único, el ModuleSpecID. El ModuleSpecID contiene el ModuleClassID (es decir, el ModuleClassID es integrado en un ModuleSpecID), indicando la clase módulo asociada.

Una especificación de módulo implica compatibilidad de red. Todas las implementaciones de una especificación de módulo dada deben usar los mismos protocolos y ser compatibles, aunque pueden escribirse en diferentes lenguajes.

6.3.4.3. Module Implementation:

La implementación del módulo es la implementación de una especificación de módulo dada. Puede haber múltiples implementaciones del módulo para una especificación de módulo dada. Cada implementación del módulo contiene el ModuleSpecID de la especificación asociada que lleva a cabo.

Los módulos son usados por los servicios de los grupos de pares, y también pueden usarse por servicios autónomos. Los servicios JXTA pueden usar la abstracción en módulos para identificar la existencia del servicio (su Clase Módulo), la especificación del servicio (su Especificación de Módulo), o una implementación del servicio (una Implementación del Módulo). Cada uno de estos componentes tiene un anuncio asociado que puede publicarse y descubrirse por otros pares de JXTA.

Como un ejemplo, el Servicio de Descubrimiento de JXTA. Tiene un único ModuleClassID, identificándolo como un servicio de descubrimiento, su funcionalidad abstracta. Puede haber múltiples especificaciones del servicio de descubrimiento, cada una posiblemente incompatible entre sí. Una especificación puede usar estrategias diferentes ajustadas al tamaño del grupo y su dispersión por la red, mientras otra experimenta con nuevas estrategias. Cada especificación tiene un único ModuleSpecID que hace referencia al ModuleClassID del servicio de descubrimiento. Para cada especificación, puede haber múltiples implementaciones cada una de los cuales contiene el mismo ModuleSpecID.

En el resumen, puede haber múltiples especificaciones de una clase módulo dada, y cada una puede ser incompatible. Sin embargo, se suponen compatibles todas las implementaciones de cualquier especificación dada.

6.3.5. Pipes

Los pares de JXTA usan pipes (tuberías) para enviarse mensajes entre si. Los pipes son un mecanismo asincrónico y unidireccional de transferencia de mensajes usado para el servicio de comunicación. Los pipes no discriminan; soportan la transferencia de cualquier objeto, incluyendo código binario, data strings, y objetos basados en tecnología Java.

Son llamados puntos finales (puntos finales) del pipe la entrada del

pipe (extremo del receptor) y la salida del pipe (extremo del emisor). Los puntos finales del pipe se ligan dinámicamente a los puntos finales del par en tiempo de ejecución. Estos puntos finales del par se corresponden con los interfaces de red de pares disponibles (por ejemplo, un puerto de TCP y dirección IP asociada) eso puede usarse para enviar y recibir el mensaje. Los pipes de JXTA pueden tener puntos finales que se conecten a diferentes pares en momentos diferentes, o no pueden no conectarse en modo alguno.

Los pipes son canales virtuales de comunicación y pueden conectar pares que no tengan un enlace físico directo. En este caso, se usan uno o más puntos finales de pares intermediarios para transmitir los mensajes entre los dos puntos finales del pipe.

Los pipes ofrecen dos modos de comunicación, punto a punto y propagación, como se ve en la **Figura 23**. El núcleo de JXTA también proporciona pipes seguros unicast, una variante segura del pipe punto a punto.

- **Pipes punto a punto**. Un pipe punto a punto interconecta exactamente dos puntos finales de pipe: una entrada de pipe en un par recibe mensajes enviados desde la salida de pipe de otro par.

- **Pipes de propagación**. Un pipe de propagación conecta una salida de pipe a múltiples entradas de pipe. Los mensajes fluyen desde la salida del pipe (el origen de la propagación) a las entradas de pipe. Toda la propagación se hace dentro del alcance de un grupo de pares. Es decir, la salida y todas las entradas deben pertenecer al mismo grupo de pares.

- **Pipes Seguros Unicast**. Un pipe seguro unicast es un tipo de pipe punto a punto que proporciona un canal de comunicación seguro.

Pueden construirse tipos de servicios de pipe adicionales usando los pipes básicos del núcleo. Por ejemplo, la plataforma actual de J2SE incluye pipes bidireccionales y pipes bidireccionales/fiables.

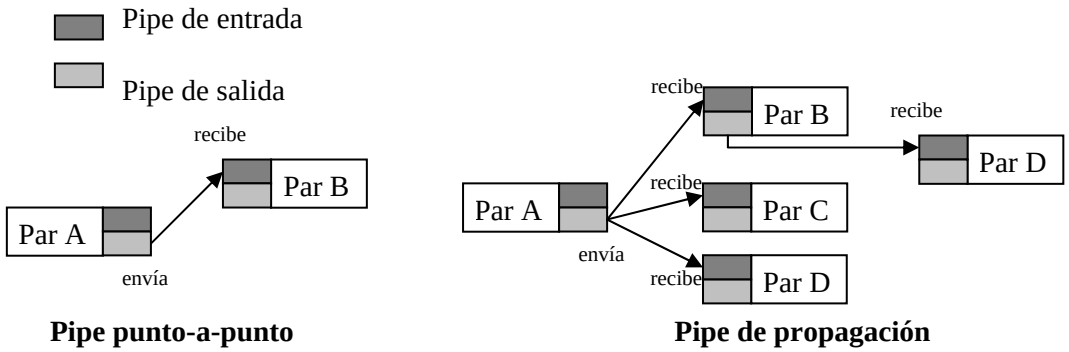


Figura 23: pipes punto a punto y de propagación.

6.3.6. Mensajes

Un mensaje es un objeto que se envía entre pares de JXTA; es la unidad básica de intercambio de datos entre los pares. Los mensajes se envían y se reciben por el Servicio de Pipe y por el Servicio de Puntos finales. Típicamente, las aplicaciones usan el Servicio de Pipe para crear, envían, y recibir los mensajes. (En general, no se espera que las aplicaciones necesiten usar el Servicio de Puntos finales directamente. Sin embargo, si una aplicación necesita entender o controlar la topología de la red de JXTA, puede usar el Servicio de Puntos finales.)

Un mensaje es una sucesión ordenada de contenidos nombrados y clasificados llamados los elementos del mensaje. Así un mensaje es esencialmente un conjunto de pares nombre/valor. El contenido puede ser un tipo arbitrario.

Los protocolos de JXTA son especificados como un conjunto de mensajes intercambiados entre pares. Cada plataforma software describe cómo un mensaje se convierte a y desde una estructura de datos nativa como un objeto Java o una estructura de C.

Hay dos representaciones para los mensajes: XML y binario. La plataforma de ligado de JXTA J2SE usa un envoltorio en formato binario para encapsular la carga útil del mensaje. Los servicios pueden usar el formato más adecuado para ese transporte (por ejemplo, un servicio que requiera una representación compacta para mensajes puede usar la representación binaria, mientras otros servicios pueden usar XML). Pueden codificarse los datos binarios usando un esquema de codificación Base64 en el cuerpo de un mensaje XML.

El uso de mensajes de XML para definir los protocolos permite a muchos tipos diferentes de pares participar en un protocolo.

Debido a que los datos se etiquetan, cada par es libre de implementar el protocolo del modo más apropiado a sus capacidades y su papel. Si un par sólo necesita algún subconjunto del mensaje, las etiquetas de datos XML le permiten a ese par que identifique las partes del mensaje que le son de interés. Por ejemplo, un par que está muy forzado y tiene capacidad insuficiente de procesar algo o la mayoría de un mensaje puede usar la etiqueta de datos para extraer las partes que puede procesar, e ignorar el resto.

6.3.7. Anuncios (Advertisements)

Todos los recursos de una red JXTA, como los pares, grupos de pares, pipes, y servicios, se representan por un anuncio. Los anuncios son estructuras de meta-datos en un lenguaje neutro representadas como documentos XML. Los protocolos de JXTA usan los anuncios para describir y publicar la existencia de recursos de un par. Los pares descubren los recursos buscando sus anuncios correspondientes, y pueden almacenar cualquier anuncio descubierto localmente.

Cada anuncio se publica con una vida que especifica la disponibilidad de su recurso asociado. Las vidas permiten la eliminación de recursos obsoletos sin requerir ningún control centralizado. Un anuncio puede ser republicado (antes de que el anuncio original caduque) para extender la vida de un recurso.

Los protocolos de JXTA definen los siguientes tipos de anuncios:

- **Peer Advertisement** (Anuncio de par). Describe el recurso del par. El uso primario de este anuncio es mantener información específica sobre el par, como su nombre, identificador de par, puntos finales disponibles, y cualquier atributo de tiempo de ejecución que los servicios de grupo individuales quieran publicar (como ser un par rendezvous (o de citas) para el grupo).

- **Peer Group Advertisement** (Anuncio de grupo de pares). describe los recursos del grupo, como el nombre, el identificador de grupo, descripción, especificación, y parámetros de servicio.

- **Pipe Advertisement** (Anuncio de pipe). describe un canal de comunicación pipe, y se usa por el servicio de pipe para crear los puntos finales del pipe de entrada y salida asociados. Cada anuncio de pipe contiene un identificador simbólico opcional, un descriptor de tipo de pipe (punto a punto, de propagación, seguro, etc.) y un identificador único de pipe.

- **Module Class Advertisement** (Anuncio de la Clase Módulo). describe una clase módulo. Su propósito primario es documentar formalmente la existencia de una clase módulo. Incluye un nombre, descripción, y un identificador único (ModuleClassID).

- **Module Spec Advertisement** (Anuncio de la especificación de Módulo). define una especificación del módulo. Su propósito principal es proporcionar las referencias a la documentación necesaria para crear implementaciones conformadas de esa especificación. Un uso secundario es, opcionalmente, hacer instancias ejecutables remotamente utilizables, publicando la información como un anuncio de pipe. Incluye nombre, descripción, identificador único (ModuleSpecID), anuncio del pipe, y campo de parámetros que contiene parámetros arbitrarios para ser interpretados por cada implementación.

- **Module Impl Advertisement** (Anuncio de la implementación del Módulo). define una implementación de una especificación de módulo dada. Incluye nombre, ModuleSpecID asociado, así como el código, el paquete, y el campo de parámetros que permite a un par recuperar los datos necesarios para ejecutar la implementación.

- **Rendezvous Advertisement** (Anuncio de Rendezvous). describe a un par que actúa como un par rendezvous para un grupo dado.

- **Peer Info Advertisement** (Anuncio de información de par). describe el recurso de información de par. El uso primario de este anuncio es mantener una información específica sobre el estado actual de un par, como el tiempo activo, la cuenta de los mensajes de entrada y de salida, hora en la que se recibió el último mensaje, y hora en la que se envió el último mensaje.

Cada anuncio se representa por un documento XML. Los anuncios están compuestos de una serie de elementos jerárquicamente organizados. Cada elemento puede contener datos o elementos adicionales. Un elemento también puede tener atributos. Los atributos son los pares de cadenas nombre-valor. Un atributo se usa para almacenar meta-datos que ayudan describir los datos dentro del elemento.

Un ejemplo de anuncio de pipe se incluye en la siguiente figura:

```
   <?xml version="1.0"?>
   <!DOCTYPE jxta:PipeAdvertisement>
   <jxta:PipeAdvertisement xmlns:jxta="http://jxta.org">
   <Id>
   urn:jxta:uuid59616261646162614E504720503250338E3E78622
9EA460DADC   1A176B69B731504
   </Id>
   <Type>
   JxtaUnicast
   </Type>
   <Name>
   TestPipe.end1
   </Name>
   </jxta:PipeAdvertisement>
```

La especificación completa de los anuncios de JXTA se da en la especificación de Protocolos JXTA. Servicios o implementaciones de par pueden subclasificar cualquiera de los anuncios anteriores para crear sus propios anuncios.

6.3.8. Seguridad

Las redes dinámicas P2P como la red JXTA necesitan soportar diferentes niveles de acceso a los recursos. Los pares de JXTA trabajan sobre un modelo basado en la confianza, en el cual un par individual actúa bajo la autoridad que le fue concedida, por otro par confiado, para realizar una tarea en particular.

Cinco requisitos básicos de seguridad deben ser provistos:

- **Confidencialidad**. Garantiza que los contenidos de un mensaje no son revelados a los individuos no autorizados.

- **Autentificación**. Garantiza que el remitente es quién él o ella afirma ser.

- **Autorización**. Garantiza que el remitente está autorizado para enviar un mensaje.

- **Integridad de los datos**. Garantiza que el mensaje no fue modificado accidentalmente o deliberadamente en el camino.

- **Refutabilidad**. Garantiza que el mensaje fue transmitido por un remitente apropiadamente identificado y no es una repetición de un mensaje previamente transmitido.

Los mensajes XML proveen la habilidad para añadir meta-datos como credenciales, certificados, compendios, y claves públicas para los mensajes JXTA, posibilitando estos requisitos básicos de seguridad para ser encontrado. Los compendios de mensaje garantizan la integridad de los datos de los mensajes. Los mensajes también pueden ser encriptados (usando claves públicas) y firmados (usando certificados) para la confidencialidad y refutabilidad. Las credenciales pueden usarse para proveer autentificación de mensaje y autorización.

Una credencial es una seña que se usa para identificar un remitente, y puede usarse para verificar el derecho que tiene un remitente para enviar un mensaje a un punto final especificado. La credencial es una seña opaca que debe ser presentada de nuevo cada vez que un mensaje es enviado. La dirección remitente colocada en una envoltura de mensaje JXTA es comprobada varias veces con la identidad del remitente en la credencial. La implementación de cada credencial es especificada como una configuración de plug-in, lo cual permite configuraciones múltiples de autentificación coexistiendo en la misma red.

Es un intento de los protocolos JXTA el ser compatible con los mecanismos de seguridad ampliamente aceptados de la capa de transporte para las arquitecturas basadas en mensajes, como la Capa de Seguridad de Sockets (SSL) y de Protocolo Internet Seguro (IPSec). Sin embargo, los protocolos seguros de transporte como SSL e IPSec sólo proveen la integridad y la confidencialidad de transferencia de mensaje entre dos pares comunicantes. Para proveer transferencia segura en una red de multisaltos como JXTA, una la asociación de confianza debe ser establecida entre todos los pares intermediarios. La seguridad es comprometida si alguno de los enlaces de la comunicación no es asegurado.

6.3.9. Identificadores

Los pares, grupos de pares, pipes y otros recursos JXTA necesitan ser excepcionalmente identificables. Un ID JXTA identifica únicamente a una entidad y sirve de método ortodoxo de referir a esa entidad. Actualmente, hay seis tipos de entidades JXTA que tienen tipos de identificadores JXTA definidos: Los pares, grupo de pares, pipes, contenidos, clases módulo, y especificaciones de módulo.

Las URNs se usan para expresar identificadores JXTA. URNs es una forma de URI que *" ... se propone servir identificadores de recurso persistentes, independientes de la situación".* Como otras clases de URI, los identificadores JXTA son presentados como texto.

Un ejemplo de identificador de par JXTA es:

```
urn:jxta:uuid-
9616261646162614A78746150325033F3BC76FF13C2414CBC0AB663666DA53903
```

Un ejemplo identificador de pipe JXTA es:

```
urn:jxta:uuid-
9616261646162614E504720503250338E3E786229EA460DADC1A176B69B731504
```

Los ID's únicos son generados aleatoriamente por la plataforma JXTA J2SE de ligado. Hay dos identificadores JXTA especiales reservados: el identificador nulo y el identificador Net Peer Group.

6.4. ARQUITECTURA DE RED.

6.4.1. Organización de la Red

La red JXTA es una red ad-hoc, multisalto, y adaptable compuesta de pares conectados. Las conexiones en la red pueden ser transitorias, y el enrutamiento de mensajes entre pares no es determinista. Los pares pueden asociarse o pueden dejar la red en cualquier momento, y las rutas pueden alterarse frecuentemente.

Los pares pueden tomar cualquier forma mientras puedan comunicarse usando los protocolos JXTA. La organización de la red no es promulgada por el esqueleto JXTA, pero en la práctica se usan típicamente cuatro tipos de pares:

6.4.1.1. Par de borde mínimo.

Un par de borde mínimo puede enviar y recibir mensajes, pero no guarda en caché anuncios o mensajes de la ruta a otros pares. Los pares en dispositivos con recursos limitados (Ej. una PDA o un teléfono celular) probablemente serán pares de borde mínimo.

6.4.1.2. Par de borde de amplio espectro.

Un par de borde de amplio espectro puede enviar y recibir mensajes, y típicamente podrá guardar en caché anuncios. Un par simple contesta a las peticiones de descubrimiento con información encontrada en sus anuncios guardados en caché, pero no reenvía cualquier petición de descubrimiento. Es probable que la mayoría de los pares sean pares de borde.

6.4.1.3. Par Rendezvous (par de citas).

Un par rendezvous es como cualquier otro par, y mantiene una memoria caché de anuncios. Sin embargo, los pares rendezvous también reenvían peticiones de descubrimiento para ayudar a otros pares a descubrir recursos. Cuando un par se une a un grupo de pares, automáticamente busca un par rendezvous. Si ningún par rendezvous fuera encontrado, dinámicamente se convierte en un par rendezvous para ese grupo de pares. Cada par rendezvous mantiene una lista de otros pares rendezvous conocidos y los pares que lo usan como rendezvous.

Cada grupo de pares mantiene su propio conjunto de pares rendezvous, y puede tener tantos pares rendezvous como necesite. Sólo los pares rendezvous que forman parte de un grupo de pares verán las peticiones específicas de búsqueda del grupo.

Los pares de borde envían las peticiones de búsqueda y de descubrimiento a los pares rendezvous, los cuales en el turno hacia adelante de peticiones no pueden contestar a otros pares rendezvous conocidos. El procedimiento de descubrimiento continúa hasta que un par tiene la respuesta o la petición muere. Los mensajes tienen un tiempo de vida predeterminado (TTL) de siete saltos. Los Loopbacks son evitados manteniendo una lista con los pares que va pasando a lo largo del camino del mensaje.

6.4.1.4. Relay peer (par de retransmisiones).

Un par Relay (par de retransmisiones) mantiene información acerca de las rutas para otros pares y encamina mensajes a los pares. Un par primero busca en su memoria caché local la información de la ruta. Si no es encontrada, entonces el par envía las consultas a los pares relay pidiendo la información de la ruta. Los pares Relay también reenvían mensajes en representación de pares que directamente no pueden direccionar a otro par (Ej., Los ambientes NAT), cruzando diferentes redes físicas y/o lógicas.

Cualquier par puede implementar los servicios requeridos para ser un relay o un rendezvous. Los servicios relay y rendezvous pueden ser implementados como una pareja en el mismo par.

6.4.2. Índice Distribuido de Recursos Compartidos.

La plataforma JXTA J2SE 2.0 da soporte a un servicio de índice distribuido de recursos compartidos (SRDI) para proveer un mecanismo más eficiente para propagar las peticiones de consulta dentro de la red JXTA. Los ambientes NAT), cruzando diferentes redes físicas y/o lógicas.

Los pares Rendezvous mantienen un índice de anuncios publicados por pares de borde. Cuando los pares de borde publican anuncios nuevos, usan el servicio SRDI para promover índices de anuncio a sus rendezvous. Con esta jerarquía de pares rendezvous de borde, las consultas son propagadas entre rendezvous sólo, que reduce significativamente el número de pares involucrados en la búsqueda de un anuncio.

Cada par rendezvous mantiene su propia lista de rendezvous conocidos en el grupo de pares. Unos rendezvous pueden recuperar información de rendezvous de un conjunto predefinido de rendezvous inicializando o preseleccionando. Los Rendezvous periódicamente seleccionan un número aleatorio dado de pares rendezvous y les envían una lista aleatoria de sus rendezvous conocidos. Los Rendezvous también purgan periódicamente los rendezvous que no contestan. Así, mantienen una red holgadamente consistente de pares rendezvous conocidos.

Cuando un par publica un anuncio nuevo, el anuncio es indexado por el servicio SRDI usando claves como el nombre del anuncio o el identificador. Sólo los índices del anuncio son promovidos a los rendezvous por SRDI, minimizando la cantidad de datos que

necesitan ser almacenados en los rendezvous. Los rendezvous también promueven el índice a pares rendezvous adicionales (seleccionados por el cálculo de una función de comprobación aleatoria del índice del anuncio).

6.4.2.1. Consultas

Un ejemplo de configuración se muestra en la **Figura 24**. El par A es un par de borde, y es configurado para usar el Peer R1 como su rendezvous. Cuando el Peer A inicia una petición de descubrimiento o de búsqueda, se envía inicialmente a su par rendezvous (R1 en este ejemplo) y también por multicast para otros pares en la misma subred.

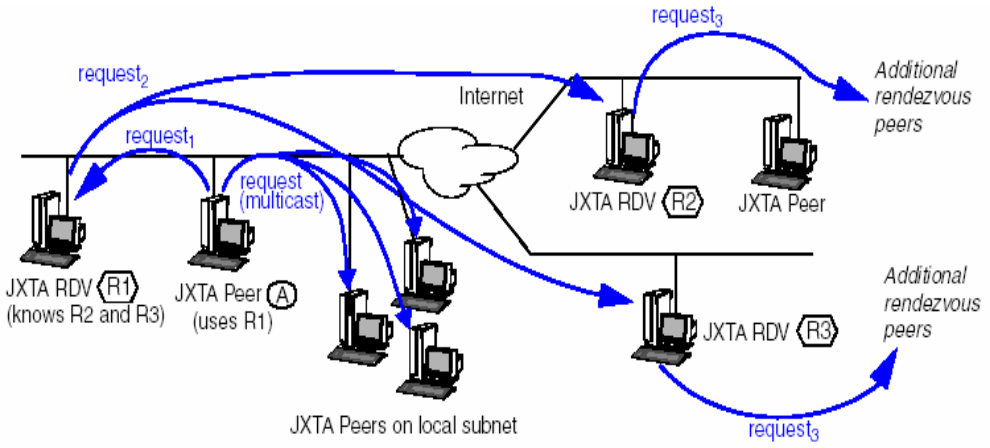


Figura 24: Propagación de Peticiones por pares rendezvous.

Las consultas locales a vecinos (i.e., dentro de una subred de comunicación) son propagadas a los pares vecinos usando lo que un transporte define como el método broadcast (difusión) o multicast. Los pares receptores de la consulta responden directamente al par emisor, si contienen la información en su memoria caché local.

Las consultas fuera de la vecindad local se envían al par rendezvous conectado. El par rendezvous trata de satisfacer la consulta buscando en su memoria caché. Si contiene la información demandada, entonces contesta directamente al par que hizo la petición y no sigue propagando la petición. Si contiene el índice para el recurso en su SRDI, entonces notificará al par que publicó el recurso y ese par responderá directamente al par que hizo la petición (los rendezvous almacenan sólo el índice para el anuncio, y no para el anuncio en sí mismo).

Si el par rendezvous no contiene la información demandada,

entonces un algoritmo predeterminado se usa para recorrer caminando el conjunto de rendezvous buscando un rendezvous que contenga el índice. Una cuenta de saltos se usa para especificar el número máximo de veces que la petición puede ser reenviada. Una vez que la consulta alcanza al par, éste contesta directamente al que originó de la consulta.

La **Figura 25** describe una vista lógica de cómo opera el servicio SRDI. El par 2 publica un anuncio nuevo, y un mensaje SRDI se envía a su rendezvous, R3. Los índices se guardarán en R3, y pueden ser promovidos a otros rendezvous en el grupo de pares. Ahora, Peer 1 envía una petición de consulta para este recurso a su rendezvous, R1. El Rendezvous R1 comprobará su memoria caché de entradas SRDI local, y propagará la consulta si no es encontrada. Cuando el recurso esté localizado en Peer 2, Peer 2 responderá directamente a P1 con el anuncio demandado.

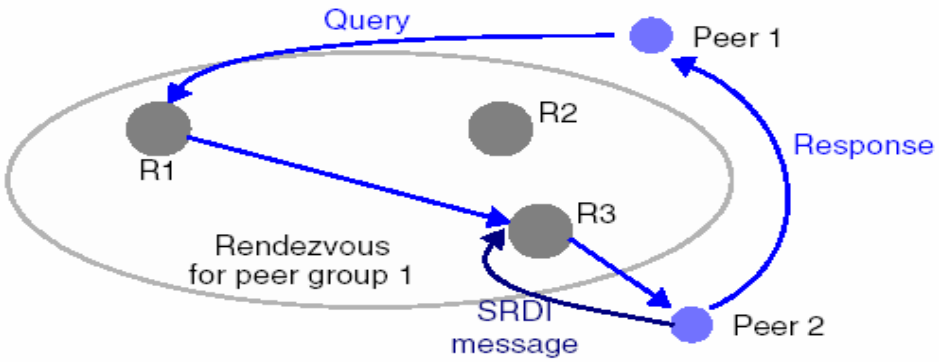


Figura 25: Distribuyendo y consultando las entradas del recurso SRDI.

6.4.3. Firewalls y NAT's.

Un par detrás de un firewall puede enviar un mensaje directamente a un par fuera de un firewall. Pero un par fuera del firewall no puede establecer una conexión directamente con un par detrás del firewall.

Para que los pares JXTA se comuniquen entre sí a través un firewall, deben darse las siguientes condiciones:

Al menos un par en el grupo de pares de dentro del firewall debe ser consciente de al menos un par de fuera del firewall.

El par interior y el par exterior del firewall deben ser conscientes del otro y deben soportar HTTP.

El firewall tiene que permitir transferencias de datos HTTP.

Estas transferencias HTTP a través del firewall no necesitan estar restringidas al puerto 80, aunque ese es el puerto usado por defecto por la mayoría de navegadores de Internet y por el actual Project JXTA J2SE binding.

La **Figura 26** describe escenario típico de encaminamiento de mensajes a través de un firewall. En este escenario, los Peers A y B quieren pasarse un mensaje, pero el firewall les impide comunicarse directamente. El Peer A primero hace una conexión a C usando un protocolo como el HTTP que puede penetrar el firewall. Luego el Peer C hace una conexión al Peer B, usando un protocolo como TCP/IP. Una conexión virtual se hace ahora entre los Peers A y B.

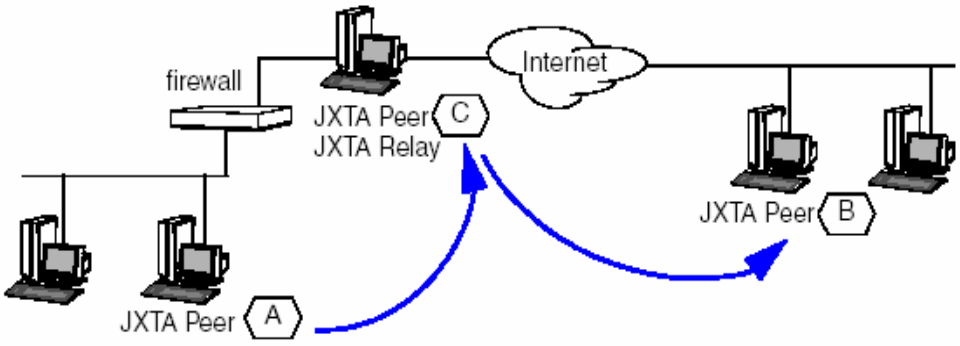


Figura 26: escenario típico de encaminamiento de mensajes a través de un firewall

6.5. PROTOCOLOS JXTA.

JXTA define una serie de formatos de mensaje XML, o protocolos, para la comunicación entre pares. Los pares usan estos protocolos para descubrirse unos a otros, anunciar y descubrir recursos de la red, y comunicarse y encaminar los mensajes.

Hay seis protocolos JXTA:

- **Protocolo de Descubrimiento de par** (Peer Discovery Protocol PDP): Usado por los pares para anunciar sus recursos (por ejemplo pares, grupos de pares, pipes, o

servicios) y descubrir recursos de otros pares. Cada recurso del par se describe y publica usando un anuncio.

- **Protocolo de Información de par** (Peer Information Protocol PIP): Usado por los pares para obtener información acerca del estado (el tiempo activo, el estado, el tráfico reciente, etc.) de otros pares.

- **Protocolo de Resolución de par** (Peer Resolver Protocol PRP): Habilita a los pares el enviar una consulta genérica a uno o más pares y recibir una respuesta (o respuestas múltiples) para la consulta. Las consultas pueden ser dirigidas a todos los pares de un grupo de pares o a pares específicos dentro del grupo. A diferencia de PDP y PIP, los cuales se usan para consultar información específica predefinida, este protocolo permite a los servicios de par definir e intercambiar cualquier información arbitraria que necesiten.

- **Protocolo de asociación de Pipes** (Pipe Binding Protocol PBP): Usado por los pares para establecer un canal de comunicación virtual, o pipe, entre uno o más pares. El PBP es usado por un par para ligar dos o más finales de la conexión (los puntos finales del pipe).

- **Protocolo de encaminamiento a puntos finales** (Endpoint Routing Protocol ERP): Usado por los pares para encontrar rutas (caminos) para los puertos de destino en otros pares. La información de la ruta incluye una secuencia ordenada de ID's de pares relay que pueden usarse para enviar un mensaje al destino. (Por ejemplo, el mensaje puede ser entregado enviándolo al par A que lo transmite al par B que lo transmite al destino final.).

- **Protocolo de Rendezvous** (Rendezvous Protocol RVP): El mecanismo por el cual los pares pueden subscribirse o pueden ser un suscriptor para un servicio de propagación. . Dentro de un grupo de pares, los pares pueden ser pares rendezvous o pares que están escuchando a pares rendezvous. el RVP permite a un par enviar mensajes a todas las instancias del servicio que estén escuchando. El RVP es usado por el Peer Resolver Protocol y el Pipe Binding Protocol para propagar mensajes.

Todos los protocolos JXTA son asincrónicos, y se basan en un modelo de consulta/respuesta. Un par JXTA usa uno de los protocolos para enviar una consulta a uno o más pares en su grupo de pares. Puede recibir cero, una, o más respuestas para su consulta.

Por ejemplo, un par puede usar PDP para enviar una consulta de descubrimiento preguntando por todos los pares conocidos en el Net Peer Group predeterminado. En este caso, probablemente contestarán múltiples pares con respuestas de descubrimiento. En otro ejemplo, un par puede enviar una petición de descubrimiento preguntando por un pipe específico llamado "aardvark". Si este pipe no es encontrado, entonces serán enviadas cero respuestas de descubrimiento en respuesta.

6.5.1. Peer Discovery Protocol (PDP).

El Protocolo de Descubrimiento de par (PDP) se usa para descubrir cualquier recurso de par publicado. Los recursos son representados como anuncios. Un recurso puede ser un par, grupo de pares, pipe, servicio, o cualquier otro recurso que tiene un anuncio.

PDP permite a un par encontrar anuncios en otros pares. . El PDP es el protocolo predeterminado de descubrimiento para todo usuario que defina grupos de pares y el net peer group predeterminado. Servicios personalizados de descubrimiento pueden escogerse para ampliar el PDP. Si un grupo de pares no tiene su propio servicio de descubrimiento, entonces el PDP se usa para sondear los pares en busca de anuncios.

Hay múltiples formas para descubrir información distribuida. La plataforma Project JXTA J2SE binding actual usa una combinación de IP multicast para la subred local y el uso de pares rendezvous, una técnica basada en networkcrawling.

Los pares Rendezvous proveen el mecanismo de enviar peticiones de un par conocido al siguiente ("arrastrándose" por la red) para descubrir información dinámicamente. Un par puede ser preconfigurado con un conjunto predefinido de pares rendezvous. Un par también puede escoger inicializarse dinámicamente localizando a pares rendezvous o recursos de la red en su entorno cercano. Otras técnicas, como redes de contenido direccionable (CAN's), también podrían agregarse para realizar descubrimiento de recursos.

Los pares generan peticiones de mensajes de consulta de descubrimiento para descubrir anuncios dentro de un grupo de pares. Este mensaje contiene la credencial del grupo de pares del par de sondeo e identifica al par de sondeo para el destinatario del mensaje. Los mensajes pueden enviarse a cualquier par dentro de una región o a un par rendezvous.

Un par puede recibir cero, una, o más respuestas para una petición de consulta de descubrimiento. El mensaje de respuesta devuelve uno o más anuncios.

6.5.2. Peer Information Protocol (PIP).

Una vez que un par está ubicado, sus capacidades y estado pueden ser consultados. El Protocolo de Información de par (PIP) provee un conjunto de mensajes para obtener información de estado de par. Esta información puede usarse para la implementación comercial o interna de aplicaciones JXTA. Por ejemplo, en las implementaciones comerciales la información puede usarse para determinar el uso de un servicio de par y facturar a los consumidores del servicio por su uso. En una implementación interna de IT, la información puede ser usada por el departamento IT para supervisar el comportamiento de un par y cambiar itinerario del tráfico de la red para mejorar el funcionamiento global.

Estos enganches pueden ser extendidos para proveer al departamento de información tecnológica control del par además de proveer información de estado.

El mensaje ping de PIP se envía a un par para comprobar si el par está vivo y para informarse sobre el par. El mensaje ping especifica si se debe devolver una respuesta total (anuncio de par) o un reconocimiento simple (si está encendido y el tiempo activo).

El mensaje de información de par se usa para enviar un mensaje en respuesta a un mensaje ping. Contiene la credencial del emisor, el identificador del par emisor y el identificador del par objetivo, el tiempo activo, y el anuncio de par.

6.5.3. Peer Resolver Protocol (PRP).

El Protocolo de Resolución de par (PRP) habilita a los pares enviar peticiones genéricas de consulta a otros pares e identifica las respuestas

encontradas. Las peticiones de consulta pueden enviarse a un par específico, o pueden ser propagadas por los servicios rendezvous dentro del alcance de un grupo de pares. El PRP usa el Servicio Rendezvous para propagar una consulta a múltiples pares, y usa mensajes unicast para enviar consultas a pares especificados.

El PRP es un protocolo fundamental que da soporte a peticiones genéricas de consulta. Ambos PIP y PDP se construyen usando PRP, y proveen consultas/peticiones específicas: El PIP se usa para consultar información específica de estado y PDP se usa para descubrir recursos de par. El PRP puede servir para cualquier consulta genérica que pueda necesitar una aplicación. Por ejemplo, el PRP permite a los pares definir e intercambiar consultas para encontrar o buscar información de servicio como el estado del servicio, el estado de un punto final de un pipe, etc.

El mensaje de consulta de resolución se usa para enviar una petición de consulta de resolución a un servicio en otro miembro de un grupo de pares.

El mensaje de consulta de resolución contiene la credencial del emisor, un identificador único de consulta, un manipulador de servicio específico, y la consulta. Cada servicio puede registrar un manipulador en el servicio de resolución del grupo de pares para procesar las peticiones de consulta de resolución y generar respuestas. El mensaje de respuesta de resolución se usa para enviar un mensaje en respuesta a un mensaje de consulta de resolución.

El mensaje de respuesta de resolución contiene la credencial del emisor, un identificador único de búsqueda, un manipulador de servicio específico y la respuesta. Se pueden enviar múltiples mensajes de consulta de resolución. Un par puede recibir cero, una, o más respuestas para una petición de consulta.

Los pares también pueden participar en el índice distribuido de recursos compartidos (SRDI). SRDI provee un mecanismo genérico, donde los servicios JXTA pueden utilizar un índice distribuido de recursos compartidos con otros pares agrupados como un conjunto de pares más capaces como los pares rendezvous. Estos índices pueden usarse para dirigir consultas en la dirección donde la consulta es más probable que sea contestada, y repropagar mensajes a los pares interesados en ésta propagación de mensajes. El PRP envía a un mensaje SRDI de resolución al manipulador nombrado en uno o más pares en el grupo de pares. El mensaje SRDI de resolución se envía a un manipulador

específico, y contiene una cadena que se interpretará por el manipulador apuntado.

6.5.4. Pipe Binding Protocol (PBP).

El Protocolo de asociación de pipe (PBP) es usado por los miembros del grupo de pares para vincular un anuncio de pipe a un punto final de pipe. El enlace virtual del pipe (el camino) puede ser estratificado en cualquier número de eslabones de transporte de red como TCP/IP.

Cada final del pipe trabaja para mantener el enlace virtual y restablecerlo, si es necesario, asociando o encontrando los puntos finales de pipe asociados actualmente.

Un pipe puede ser visto como una cola mensajes abstracta y nombrada, soportando las operaciones de crear, abrir/resolver (asociar), cerrar (desligar), suprimir, enviar, y recibir. Las implementaciones efectivas de pipe pueden diferir, pero todas las implementaciones conformes usan a PBP para asociar el pipe a un punto final. Durante la operación abstracta de creación, un par local asocia un punto final de pipe a un pipe de transporte.

El mensaje de consulta PBP se envía por un punto final de un pipe para encontrar un punto final de pipe ligado al mismo anuncio de pipe.

El mensaje de consulta puede pedir información no obtenida de la memoria caché. Esto se usa para sacar la información más actualizada de un par. El mensaje de consulta también puede contener a un identificador de par optativo, que si está presente indica que sólo el par especificado debe responder a la consulta.

Un mensaje de respuesta PBP se devuelve al par que hizo la petición por cada par ligado al pipe. El mensaje contiene el identificador de Pipe, el par donde la entrada de pipe correspondiente haya sido creada, y un valor booleano indicando si la entrada de pipe existe en el par especificado.

6.5.5. Endpoint Routing Protocol (ERP).

El protocolo de encaminamiento de Puntos finales (ERP) define un conjunto de mensajes de petición/consulta que se usan para encontrar

información de encaminamiento. Esta información de la ruta es necesaria para enviar a un mensaje de un par (la fuente) a otro (el destino).

Cuando un par recibe instrucciones de enviar un mensaje a una dirección de un punto final de par dado, primero mira en su memoria caché local para determinar si tiene una ruta para este par. Si no encuentra una ruta, entonces envía una petición de consulta de resolución de ruta a sus pares relays disponibles pidiendo información de la ruta. Cuando un par relay recibe una consulta de ruta, éste comprueba si conoce la ruta. Si esto ocurre, entonces devuelve la información de ruta como una enumeración de saltos.

Cualquier par puede consultar a un par relay por la información de la ruta, y cualquier par en un grupo de pares puede convertirse en un relay. Los pares relays típicamente guardan en caché la información de las rutas.

La información de la ruta incluye el identificador de par de la fuente, el identificador de par del destino, un tiempo de vida (TTL) para la ruta, y una secuencia ordenada de identificadores de par de las puertas de enlace (gateways). La secuencia de los identificadores de par no puede estar completa, pero debería contener al menos al primer relay.

Las peticiones de consulta de la ruta son enviadas por un par a un par relay para solicitar información de la ruta. La búsqueda puede indicar una preferencia para evitar el contenido de la memoria caché del encaminador y buscar dinámicamente una ruta nueva.

Los mensajes de respuesta de ruta se envían por un par relay en respuesta a unas peticiones de información de ruta. Este mensaje contiene el identificador de par del destino, el identificador de par y el anuncio de par del encaminador que conoce una ruta para el destino, y una secuencia ordenada de uno o más relays.

6.5.6. Rendezvous Protocol (RVP).

El Protocolo de Rendezvous (RVP) es el responsable de propagar los mensajes dentro de un grupo de pares. Mientras que los grupos de pares diferentes pueden tener diferentes métodos para propagar los mensajes, el Protocolo Rendezvous define un protocolo simple que permite:

- A los pares conectarse al servicio (puede propagar mensajes y recibir mensajes propagados).

- Controlar la propagación del mensaje (el tiempo de vida, la detección de loopbacks, etc.).

El RVP es usado por el Protocolo de Resolución de par y por el Protocolo de Asociación de Pipes para propagar los mensajes.

6.6. JXTA EN JAVA

JXTA es independiente del lenguaje de programación y del protocolo de transporte. Sin embargo, la implementación realizada en Java para la J2SE se ha convertido en la referencia del proyecto, debido fundamentalmente a 2 razones:

- Permite el rápido desarrollo y prueba de las ideas, resultando en una implementación robusta.

- Java está disponible para todo tipo de plataformas (Unix, Windows, MacOS X, etc).

Otros subproyectos de JXTA están actualmente trabajando en implementaciones de los protocolos JXTA para C/C++, Python, J2ME, etc. La mayoría de subproyectos dedicados a aplicaciones y servicios JXTA, han elegido la implementación en Java como medio para el desarrollo.

Especialmente interesante es la implementación del protocolo PDP en J2SE: usa una combinación de multidifusión IP dentro de la red local y los *peers rendezvous*. Los *peers rendezvous* proporcionan el mecanismo de envío de peticiones de un *peer* conocido a otro para descubrir información dinámicamente. Un *peer* puede ser preconfigurado con una lista de *peers rendezvous*, o puede descubrirlos en tiempo de ejecución. Otras técnicas de descubrimiento, como la implementación CAN de DHT, también pueden ser usadas. Se incluye en la versión 2.0 un algoritmo basado en índices distribuidos de recursos compartido (SRDI).

También se implementan nuevos tipos de tuberías: bidireccionales y bidireccionales confiables, extendiendo el concepto de tubería unidireccional proporcionada por el núcleo de JXTA.

El actual material de referencia disponible para usar JXTA en J2SE consta de:

- Guía del programador Java para la versión 2.0 de JXTA.
- Colección de ejemplos-guía.
- Archivos .jar de la implementación J2SE.

En la web de JXTA se encuentra todo el material relacionado con JXTA, además de todos los subproyectos relacionados y amparados por JXTA y desarrollados en Java.

7. APLICACIÓN DESARROLLADA.

7.1. INTRODUCCIÓN.

En este capítulo se pretende analizar de forma exhaustiva la aplicación que se ha desarrollado en este Proyecto Fin de Carrera. Para ello en un principio se analizarán en detalle las clases de las que se compone la aplicación y a continuación se pasa a explicar las diferentes funcionalidades que encuentra el usuario al ejecutar dicho programa.

7.2. ANALISIS DE LAS CLASES DESARROLLADAS

A continuación pasaremos a explicar desde un punto de vista más centrado en el código el funcionamiento de las distintas clases que forman nuestro programa.

Cabe destacar que esta se trata de una aplicación pensada para funcionar en una red de pares y como tal se pueden distinguir 2 partes claramente diferenciadas en nuestro programa: el cliente y el servidor, implementadas en las clases ClienteServiciosJXTA y ServidorServiciosJXTA.

7.2.1. Clase Comercio

Se trata de la clase principal del programa, donde se encuentra la función "main". Las principales funciones de esta clase son:

- Inicializar los componentes del entorno grafico de nuestra aplicación.
- Conectarse a la red JXTA, a través de una instancia de la clase PlataformaJXTA.
- Anunciar en la red JXTA todos los productos anunciados en sesiones anteriores y que todavía no han sido vendidos.
- Mostrar en los distintos paneles de la aplicación los productos anunciados, vendidos y comprados en sesiones anteriores.

- Escucha los eventos que se generan al pulsar los botones "Anunciar" y "Buscar" y al seleccionar una fila de cualquiera de las tablas de nuestro programa.

- Por otra parte esta clase se encarga también de determinar el código de producto que se le asocia a cada producto anunciado. Se trata de un código numérico que coincidirá también con el nombre del fichero del producto, seguido de la extensión ".xml". Este código nos servirá para poder identificar de forma única los diferentes productos anunciados en nuestra aplicación.

  Para determinar este código cada vez que se inicia el programa habrá que tener en cuenta el código de los productos anunciados y vendidos en sesiones anteriores, para de esta forma continuar con los códigos de los productos de la nueva sesión.

Los principales métodos de esta clase son:

`private void BotonAnunciarActionPerformed (ActionEvent evt).` Método llamado al pulsar el botón "Anunciar" y se encarga de recoger toda la información introducida por el usuario en dicho panel. Con dicha información se construye el objeto ProductoXML, que será el que genere el fichero XML y se llama a la clase Anunciar que se encargara de anunciar dicho servicio.

`private void BotonBuscarActionPerformed (ActionEvent evt).` Método invocado al pulsar el botón "Buscar" del panel con el mismo nombre. Se encarga de recoger el nombre del producto buscado introducido por el usuario y de instanciar un objeto de la clase Buscar que será la que se encargue de buscar el servicio con ese nombre.

`private void MostrarProductoVendido (MouseEvent evt)`
`private void MostrarProductoComprado (MouseEvent evt)`
`private void MostrarProductoAnunciado(MouseEvent evt)`
`private void MostrarProductosEncontrados(MouseEvent evt)`Métodos invocados al seleccionar una de las filas de las tablas de los distintos paneles de la aplicación. En función de la tabla en que nos encontremos se llamará a uno u otro método. Estos métodos son los responsables de que al seleccionar una de las filas se nos muestre el fichero XML completo del producto seleccionado, mediante un objeto de la clase MuestraProducto.

7.2.2. Clase PlataformaJXTA

La principal función de esta clase es la conexión a la red JXTA y dentro de esta al grupo de peers especifico al que se conectaran todos los usuarios de "Comercio Electrónico 1.0", donde podrán anunciar y buscar aquellos productos que les interese.

Los métodos principales de esta clase son:

- **`void startJXTA().`** Método que se encarga de conectarse a la red JXTA y dentro esta al grupo de pares por defecto "NetPeerGroup" ya que recordemos que en la tecnología JXTA las búsquedas de anuncios se realizan dentro de un determinado grupo.

- **`public boolean conectarConRendezvous(int intentos, int tiempo):`** Espera "intentos" veces un intervalo "tiempo" segundos comprobando si este objeto esta conectado a un nodo Rendezvous. Este método no conecta con un nodo Rendezvous, sino sólo comprueba que está conectado. Puede ser de utilidad para esperar un cierto tiempo a una conexión de Rendezvous, y actuar de cierta forma si no conecta después de ese tiempo.

- **`public PeerGroup entrarEnGrupo(String nombre, String descripcion):`** Realiza los trámites para entrar a forma parte de un grupo de peers JXTA especificado a partir de su nombre. Si el grupo no existe, este método lo crea a partir del nombre y la descripción especificados y se une a él. Devuelve una instancia del grupo al que se ha entrado. Este método puede tardar tiempo en completarse, puesto que realiza una búsqueda de grupos por la red JXTA. Toda la publicación y búsqueda de anuncios de productos en esta aplicación se realizarán en el grupo de peers llamado "ComercioElectronico1.0".

- **`void borrarAnuncios(LinkedList listaAnuncios)`**: Con él se borra de la caché local todos los anuncios encontrados previamente, lamentablemente se hace uso del método *void flushAdvertisement(Advertisement adv)* de la clase DiscoveryService que no funciona correctamente ya que los anuncios una vez borrados vuelven a ser encontrados.

7.2.3. Clase Anunciar

Esta case se instancia cada vez que se pulsa el botón "Anunciar" del panel con el mismo nombre. Las principales funciones de esta clase son:

- Anunciar un determinado producto, para lo cual se llama a la clase ServidorServiciosJXTA, a la que se le pasa el nombre del fichero del producto que queremos anunciar y que será el encargado de escuchar las posibles peticiones de los clientes que soliciten nuestro producto. Recordemos que se instancia un objeto de la clase ServidorServiciosJXTA por cada producto que queremos anunciar.

- Anunciar los productos que se anunciaron en sesiones anteriores y que todavía no han sido vendidos, para ello se debe leer al iniciarse el programa los fichero XML de productos que hay en el directorio "Anuncios", y anunciar uno a uno dichos productos.

7.2.4. Clase ServidorServiciosJXTA

Todo "peer" tiene dos partes, una parte cliente y otra servidor. Esta clase trata de implementar la parte servidor de todo peer.

Cada vez que se anuncia un nuevo producto, esta clase es la encargada de crear un nuevo servicio, cuyo nombre es el nombre del producto que estamos anunciando, y publica su anuncio en la red JXTA.

Por cada producto anunciado se crea un nuevo hilo que estará esperando de forma indefinida mientras esté en ejecución el programa o hasta que dicho producto se venda.

Esta clase está a la espera de que un cliente solicite nuestro servicio anunciado, en este momento se crea una conexión bidireccional entre el cliente y el servidor a través de la cual ambos se intercambian mensajes JXTA que encapsularan a los mensajes cXML.

Por tanto las principales funciones de esta clase son:

- Anunciar un servicio, asociado a un determinado fichero de producto XML, en la red JXTA

- Esperar a un cliente que este buscando dicho producto.

- Establecer una conexión bidireccional con dicho cliente.

- Enviarle en el momento de establecer la conexión el fichero del producto XML correspondiente al servicio que está anunciado dicho objeto.

- Recibir por la tubería bidireccional y discriminar los mensajes cXML ProfileRequest y OrderRequest y responder con los mensajes ProfileResponse y OrderResponse.

- Crear un objeto de la clase Vender que es la encargada de rellenar y enviar dichos mensajes.

- Borrar el anuncio y parar el hilo una vez que el producto se ha vendido.

7.2.5. Clase Vender

Esta clase interactúa con la clase ServidorServiciosJXTA, ya que al recibir esta ultima un mensaje de petición de compra se instancia un objeto de la clase Vender.

Cabe destacar los siguientes métodos de esta clase:

- **`void enviarRespuestaPerfil():`** Este método crea un objeto de la clase RespuestaCXML y rellena el documento cXML, ProfileResponse. Una vez relleno lo envía a través de la conexión bidireccional que estableció el objeto ServidorServiciosJXTA.

- **`void enviarRespuestaCompra():`**Este método crea un objeto de la clase RespuestaCXML y rellena el documento cXML, OrederResponse. Una vez relleno lo envía a través de la conexión bidireccional que estableció el objeto ServidorServiciosJXTA.

- **`void ventaRealizada(String nombreFichero):`** Este método se encarga de eliminar de la tabla del panel de

productos anunciados, la fila correspondiente al producto vendido, para pasar a añadirla al panel de productos vendidos. también se encargará este método de borrar el fichero asociado a este producto del directorio "Anuncios", para pasarlo al directorio "Vendidos".

7.2.6. Clase Buscar

Esta clase se instancia cada vez que se pulsa el botón "Buscar" del panel con el mismo nombre. Se encarga de:

- Borrar la tabla de productos encontrados en búsquedas anteriores cada vez que se pulsa dicho botón.

- Borrar los ficheros del directorio "Encontrados", que se corresponde con los productos encontrados en búsquedas anteriores.

- Llamar a la clase ClienteServiciosJXTA a la que se le pasa el nombre del producto a buscar introducido por el usuario y que será la clase encargada de buscar dicho servicio en la red JXTA.

7.2.7. Clase ClienteServiciosJXTA

Con esta clase se implementa la parte cliente que podemos encontrar en todo "peer".

Se instancia un objeto de esta clase cada vez que se pulsa el botón "Buscar", del panel con el mismo nombre.

Esta clase hace uso de los métodos getRemoteAdvertisement y getLocalAdvertisement de la clase DiscoveryService de JXTA, que trata de buscar servicios cuyo nombre coincida, al menos en parte, con la cadena introducida por el usuario en el panel "Buscar".

Por cada anuncio que encuentra crea un objeto de la clase TuberiaCliente que será el encargado de establecer una conexión bidireccional con el servidor. De esta forma se crearán tantas conexiones como anuncios se encuentren.

7.2.8. Clase TuberiaCliente

Como hemos dicho anteriormente se crea un objeto de esta clase por cada uno de los anuncios encontrados de un determinado servicio. Las principales funciones de esta clase son:

- Establecer una conexión bidireccional con el servidor del anuncio que estamos buscando.

- Recibir el fichero XML del producto que estamos buscando a través de esa conexión.

- Almacenar dichos fichero en el directorio "Encontrados", determinando previamente el nombre que le corresponde al fichero. A cada fichero se le asigna un nombre numérico empezando en 1 y de forma ascendente, de forma que cada vez que se recibe un nuevo fichero habrá que leer de dicho directorio para determinar el nombre del fichero.

- Recibir los mensajes cXML, ProfileResponse y OrderResponse, encapsulados en mensajes JXTA que se transmiten a través de la tubería bidireccional.

- Transmitir un mensaje de petición de compra tras recibir la respuesta de perfil y comprobar que todo es correcto. Será la clase Comprar la encarga de rellenar y enviar dicho mensaje OrderRequest.

7.2.9. Clase Comprar

Una vez que, gracias a la clase TuberiaCliente, se muestran todos los productos encontrados en la tabla del panel "Buscar", como en otras ocasiones el usuario seleccionando una fila verá desplegado un nuevo panel con el contenido completo del fichero XML correspondiente a ese producto. En este caso además del botón "Salir", aparece el botón "Comprar".

Al pulsar el botón "Comprar", se instancia la clase DatosComprador y se solicita al usuario que introduzca los datos de compra. En el momento que se introducen esos datos y se pulsa el botón "Comprar" de este nuevo panel es cuando se instancia un objeto de la clase Comprar.

Esta clase interactúa en todo momento con la clase TuberiaCliente, ya que se va a tener que comunicar con la parte servidora, la que anuncia el producto, y para ello usará la tubería bidireccional que establece esta clase.

Las principales funciones por tanto de la clase Comprar son:

- Rellenar y enviar la Petición de Perfil propia del protocolo cXML a través de la conexión bidireccional.

- Enviar la petición de Compra al peer que está anunciando el producto a través de la tubería establecida con este.

- Guardar el archivo cXML de Petición de Compra en el directorio "PeticionesDeCompraEnviadas", para de esta forma tener una prueba de la compra realizada.

- Guardar el archivo XML del producto comprado el directorio "Comprados", para tener almacenados todos los datos del producto comprado.

- Es importante decir que el nombre tanto del producto comprado como de su correspondiente petición de Compra es el mismo, y sigue una numeración como en casos anteriores que empieza en 1 y se incrementa en una unidad sucesivamente.

7.2.10. Clase DatosDelComprador

Como ya hemos mencionado anteriormente esta clase se instancia en el momento de pulsar el botón "Comprar" que aparece en el panel que muestra los datos completos del producto encontrado que seleccionamos.

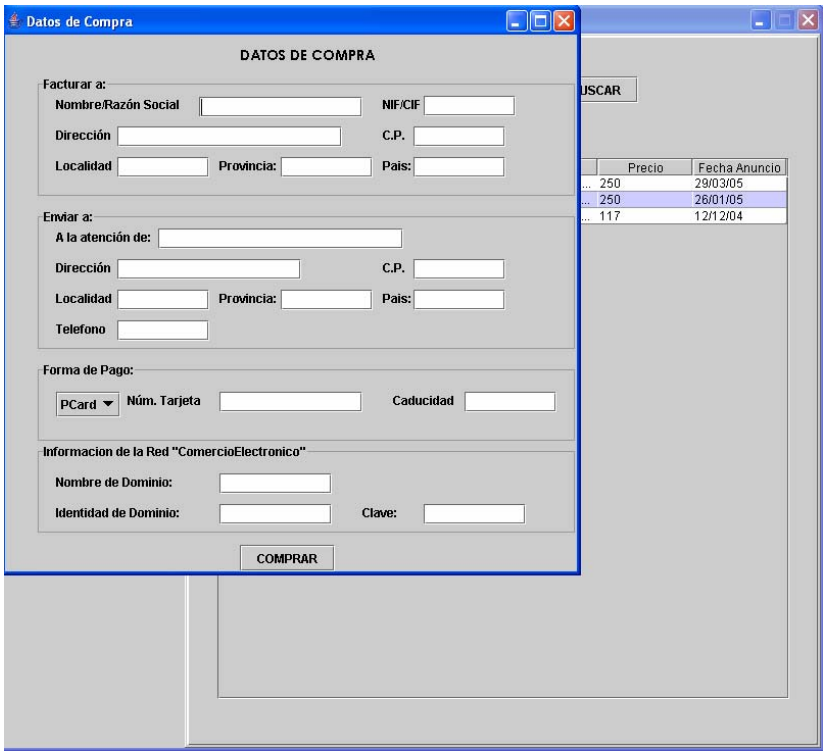
Esta clase hereda de la clase JFrame del paquete Swing. Es la responsable del panel que recoge todos lo datos de compra:

- Datos de facturación

- Datos de envío

- Datos relativos a la forma de pago

- Datos relativos a la red "Comercio Electrónico", que son datos de identificación de usuarios dentro del protocolo cXML



Todos estos datos son recogidos al pulsar el botón "Comprar" y con ellos se rellena el documento cXML correspondiente a la petición de compra, PurchaseOrder.

Una vez hecho esto se instancia un objeto de la clase Comprar al que se le pasa dicha petición de compra y se cierra automáticamente esta ventana.

7.2.11. Clase Categorías

Como ya hemos dicho todo producto que anunciemos o busquemos en nuestra red de "Comercio Electrónico 1.0", va a pertenecer a una categoría y dentro de esta a una subcategoría.

En el archivo "categorías.xml", que se suministra con el software es donde quedan recogidas las distintas categorías y subcategorías entre las que el usuario puede escoger para clasificar su producto.

Una opción posible, aunque no se implementa en este proyecto, es que dicho archivo sea transmitido por una entidad central en el momento de iniciar la aplicación, con el objeto de que el número de categorías y subcategorías no sea algo estático sino que pueda ir variando a lo largo del tiempo.

La clase Categorías va a ser la responsable de leer dicho archivo y extraer del mismo las categorías y su correspondientes subcategorías.

7.2.12. Clase ModeloTabla

Esta clase implementa la clase abstracta TableModel y va a ser superclase de otras cuatro: ModeloTablaAnunciados, ModeloTablaEncontrados, ModeloTablaVendidos, ModeloTablaComprados.

Estas clases serán las responsables de mostrar los distintos productos en las distintas tablas. Ha sido necesario generar un modelo distinto para cada tabla ya que en cada ocasión los datos que nos interesa mostrar del producto son diferentes.

Esta superclase va a implementar los métodos abstractos de TableModel además de los métodos comunes a todas sus subclases:

- **`public int insertaProducto (ProductoXML nuevoProducto)`**: Insertar un determinado producto de la tabla, para ello se hace uso de una lista enlazada LinkedList.

- **`public void borraProducto (int fila)`:** Borrar un determinado producto de la tabla.

- **`void BorraTabla()`:** Borrar la tabla al completo.

- **`void MuestraProductosAntiguos(String rutaDirectorio)`:** Al iniciar la aplicación muestra todos los productos de sesiones anteriores que se encuentran en un determinado directorio.

- **`private void avisaSuscriptores (TableModelEvent evento)`:** Cada vez que se inserta o se borra un producto de la tabla debemos lanzar un evento para que dicha

modificación se refleje en el entorno gráfico.

7.2.13. Clase ModeloTablaAnunciados

Clase que se usa como modelo para la tabla de productos anunciados, cada vez que se anuncia un nuevo producto, se enlaza el objeto ProductoXML en una lista enlazada y se presenta al usuario en la tabla.

Los atributos del producto extraídos del fichero XML que se presentan al usuario son:

- Código del producto
- Categoría del producto
- Subcategoría del producto
- Nombre del producto
- Precio
- Fecha del anuncio

7.2.14. Clase ModeloTablaEncontrados

Esta clase servirá de modelo para la tabla de productos encontrados. La lista enlazada de productos se borra en cada búsqueda, de forma que solo aparezcan en la tabla los productos correspondientes a la última búsqueda.

Los atributos que nos interesaba presentar de los productos encontrados son:

- Categoría del producto.
- Subcategoría del producto.
- Nombre del producto.

- Proveedor.
- Precio.
- Fecha del anuncio.

7.2.15. Clase ModeloTablaVendidos

Análogamente a las clases anteriores, esta será el modelo de la tabla de productos vendidos. Una vez que se realiza la venta de un determinado producto, se hará uso de esta clase para añadir el producto vendido a la lista enlazada correspondiente y para que sea mostrado en la tabla de productos vendidos.

Los atributos del producto que se muestran en esta tabla son:

- Código del producto
- Categoría del producto
- Subcategoría del producto
- Nombre del producto
- Precio

7.2.16. Clase ModeloTablaComprados

Al igual que las clases anteriores, esta servirá de modelo para la tabla de de productos comprados. En este caso los atributos del producto que se extraen del fichero XML para presentarlos al usuario son:

- Categoría del producto
- Subcategoría del producto
- Nombre del producto

- Proveedor

- Precio del producto

7.2.17. Clase MuestraProducto

Se trata de una subclase de JFrame del paquete SWING, es la clase responsable de que al seleccionar cualquier fila de cualquiera de las tablas de nuestra aplicación se despliegue un panel donde podamos ver al completo el fichero XML del producto seleccionado.

Consta de un objeto JTextArea que se incluye en un objeto JScrollPane y sobre el que se vuelca la información del objeto ProductoXML en forma de String.

7.2.18. Clase MuestraProductoEncontrado

Como hemos dicho la clase MuestraProducto se instancia cada vez que seleccionamos una fila de cualquier tabla de nuestra aplicación, a excepción de la tabla de productos encontrados. Para esta tabla se instancia un objeto de la clase MuestraProductoEncontrado que es subclase de la anterior.

La funcionalidad es la misma, mostrar al completo el fichero XML asociado al producto que seleccionamos. Pero si observamos aparece un botón más que en el panel de MuestraProducto, se trata del botón "Comprar".

Al pulsar este botón se instancia un objeto de la clase DatosComprador y se muestra dicho panel a la vez que se cierra el panel que muestra el producto.

7.2.19. Clase DocumentoXML

Esta clase representa a un documento de contenido XML de cualquier tipo. El documento se representa internamente como un objeto JDOM.

Contiene, entre otros, métodos para:

- Guardar en archivo

- Comprobar si está bien formado, y si es válido.

- Obtener valores de elementos y atributos mediante expresiones XPath.

- Crear documentos desde cero, o a partir de un objeto JDOM.

- Convertir en objeto `InputStream`

Sirve además como superclase para cualquier clase que implemente un tipo de documento XML en concreto, como documentos cXML.

7.2.20. Clase ProductoXML

Se trata de una subclase de la clase DocumentoXML, por lo tanto posee todos sus métodos.

Se trata de una clase fundamental en nuestra aplicación pues todo el manejo de productos se realiza mediante objetos de esta clase. Al instanciar esta clase se genera un objeto JDOM en el que se almacena toda la información relativa de un producto, es decir, datos del vendedor, datos de la red "Comercio Electrónico 1.0" y datos del producto.

La estructura típica de un fichero XML para un producto es:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Producto>
-    <DatosProveedor>
        <Nombre> </Nombre>
        <Apellidos> </Apellidos>
        <DNI />
        <Telefono />
        <Empresa />
        <CIF />
        <Direccion />
        <Localidad />
        <Provincia />
        <CP />
     </DatosProveedor>
-    <DatosProducto>
        <CodigoProducto></CodigoProducto>
        <Categoría> </Categoría>
        <Subcategoría> </Subcategoría>
        <NombreProducto> </NombreProducto>
        <Descripcion />
        <Precio></Precio>
        <FechaAnuncio />
     </DatosProducto>
-    <DatosRed>
        <Identity> </Identity>
        <Domain> </Domain>
     </DatosRed>
  </Producto>
```

Al llamar al constructor de esta clase se establece la estructura del objeto JDOM, pero para introducir los datos en dicho objeto se hace uso de las clases:

- `public void RellenarDatosProveedor (String nombre,String apellidos, String direccion, String localidad, String provincia, String cp, String telefono, String dni, String cif, String empresa)`

- `public void RellenarDatosProducto(String codigoProd,String categoria, String subcategoria, String nombreProducto, String descripcion, String precio, String fechaAnuncio)`

- `public void RellenarDatosRed(String identidad,`

```
String dominio)
```

Otra forma de llamar al constructor de esta clase es pasándole el fichero XML con los datos del producto, de esta forma se genera el objeto JDOM y se extra el toda la información necesaria del fichero.

Es bastante útil el método **`public String toString()`** mediante el cual el objeto JDOM se convierte en un flujo XML en forma de String. Esto se usará para transmitir estos objetos en mensajes JXTA, a través de las tuberías bidireccionales que se crean entre cliente y servidor.

7.2.21. Clase DocumentoCXML

Representa a un documento genérico usado en cXML. En principio, está pensado para documentos de tipo cXML, peticiones y respuestas de las transacciones.

Todos estos documentos se validan respecto al DTD de cXML. Esta clase hereda de `DocumentoXML`, por lo que posee todos sus métodos.

7.2.22. Clase PeticionCXML

Esta clase representa un documento cXML de tipo petición, por lo que su elemento raíz es "`cXML`". Hereda de la clase `DocumentoCXML`.

Proporciona la estructura básica de una petición cXML, además de métodos para establecer los valores de los elementos y atributos más importantes. También inicializa algunos elementos a valores por defecto.

7.2.23. Clase PeticionDeCompraCXML

Se trata de una subclase de la clase PeticionCXML, por lo que poseerá todos sus métodos.

Así como con la clase PeticionCXML establece la cabecera de la petición, que es genérica para todas las peticiones en cXML, esta clase establece el cuerpo específico de la petición de compra, OrderRequest. Posee también los métodos necesarios para completar todos los campos de la petición de compra.

La información para completar la petición de compra se extrae

por una parte de los datos introducidos por el usuario en el panel de "Datos del Comprador", que es mostrado por la clase DatosComprador, y por otra del propio fichero XML del producto que se desea comprar.

7.2.24. Clase PeticionDePerfilCXML

Esta clase al igual que la anterior es subclase de PeticionCXML y también se encarga de establecer el cuerpo específico de una petición de perfil en cXML, ProfileRequest. En este caso el cuerpo de la petición no tiene ningún dato que haya que introducir por parte del usuario, no así la cabecera que debe tomar los datos al igual que la petición de compra del panel de "Datos del Comprador" y del fichero XML del producto a comprar.

7.2.25. Clase RespuestaCXML

Esta clase representa un documento cXML de tipo respuesta, por lo que su elemento raíz es "`cXML`". Hereda de la clase `DocumentoCXML`.

Proporciona la estructura básica de una respuesta cXML, además de métodos para establecer los valores de los elementos y atributos más importantes. También inicializa algunos elementos a valores por defecto.

7.2.26. Clase FactoriaAleatoria

Esta clase implementa un generador de números y cadenas de caracteres numéricos aleatorias.

Su utilidad principal es en la generación de cadenas aleatorias en las peticiones cXML, en los elementos "`cXML`" y "`OrderRequestHeader`".

7.2.27. Clase FechaISO8601

Esta clase contiene una colección de métodos de manejo de fechas en formato ISO 8601. Se emplea en los documentos cXML

7.2.28. Clase UtilesDirectorio

Esta clase sólo contiene un método de borrado recursivo de un directorio. Elimina todos los archivos y subdirectorios del directorio especificado.

Se emplea como parche para un defecto de JXTA. En la clase de JXTA `DiscoveryService` se proporcionan métodos de borrado de anuncios en caché. Desgraciadamente, los métodos no funcionan, por lo que se ha optado por borrar el subdirectorio de "`.jxta`" empleado como caché de anuncios. Esta solución se usa en el método `limpiarCache()` de la clase `PlataformaJXTA`, que utiliza el método de esta clase para dicho borrado.

7.2.29. Clase Fichero

Esta clase presenta dos métodos fundamentalmente relacionados con el uso de ficheros:

- **`public static String LeeFichero(String nombreFichero):`** Este método lee un determinado fichero cuyo nombre se le pasa por argumento y lo transforma en un String.

- **`public static void EscribeFichero(String nombreFichero, String texto):`** Método que crea un fichero de nombre "nombreFichero" cuyo contenido se le pasa como argumento en forma de String, "texto".

7.3. FUNCIONALIDADES DE LA APLICACIÓN.

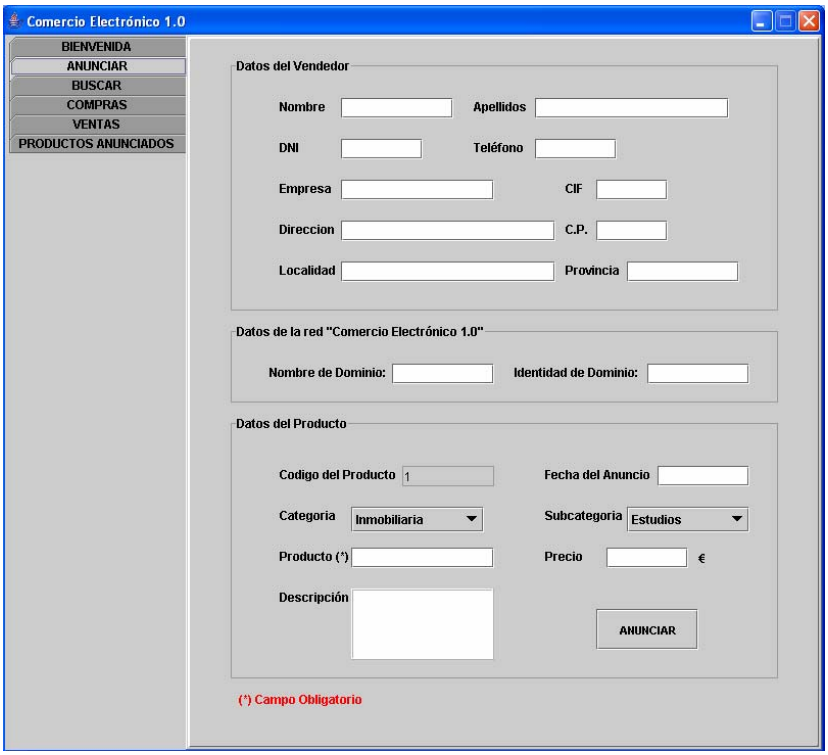
Al ejecutar el programa el usuario es recibido con una pantalla de bienvenida a la aplicación de "Comercio Electrónico 1.0", en este momento ya se habrá conectado a la red JXTA y dentro de ésta a un determinado grupo de pares, todos los usuarios de esta aplicación deben estar en el mismo grupo, ya que las búsquedas y anuncios se realizan dentro del grupo al que esté conectado el usuario.

A partir de ese momento podrá elegir entre las distintas funcionalidades del programa, pinchando en una de las otras 5 pestañas que encuentra en el lateral izquierdo de la ventana: "Anunciar", "Buscar", "Compras", "Ventas" y "Productos Anunciados".

Expliquemos a continuación el funcionamiento de las distintas acciones posibles en el programa.

7.3.1. Anunciar un producto.

El usuario podrá rellenar un formulario con los datos del producto, del vendedor, y los datos de lo que llamamos la Red "Comercio Electrónico", que no es más que una identidad y un dominio de identidad necesarios para realizar transacciones de comercio electrónico mediante el estándar cXML.



Todo producto debe pertenecer a una categoría y subcategoría predeterminadas que el usuario debe escoger. Estas categorías y subcategorías de productos permanecen en un fichero XML local que se suministra con el software denominado "categorias.xml", aunque existe la posibilidad de solicitar este fichero a un servidor central cada vez que el usuario se conecte a la red de "Comercio Electrónico P2P". La categoría y subcategoría se usan para facilitar posteriormente la búsqueda del producto en la red JXTA.

Al pulsar el botón "Anunciar" se recoge toda la información introducida y se vuelca sobre un fichero XML con una estructura como la que sigue:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Producto>
-    <DatosProveedor>
        <Nombre> </Nombre>
        <Apellidos> </Apellidos>
        <DNI />
        <Telefono />
        <Empresa />
        <CIF />
        <Direccion />
        <Localidad />
        <Provincia />
        <CP />
     </DatosProveedor>
-    <DatosProducto>
        <CodigoProducto></CodigoProducto>
        <Categoría> </Categoría>
        <Subcategoría> </Subcategoría>
        <NombreProducto> </NombreProducto>
        <Descripcion />
        <Precio></Precio>
        <FechaAnuncio />
     </DatosProducto>
-    <DatosRed>
        <Identity> </Identity>
        <Domain> </Domain>
     </DatosRed>
  </Producto>
```

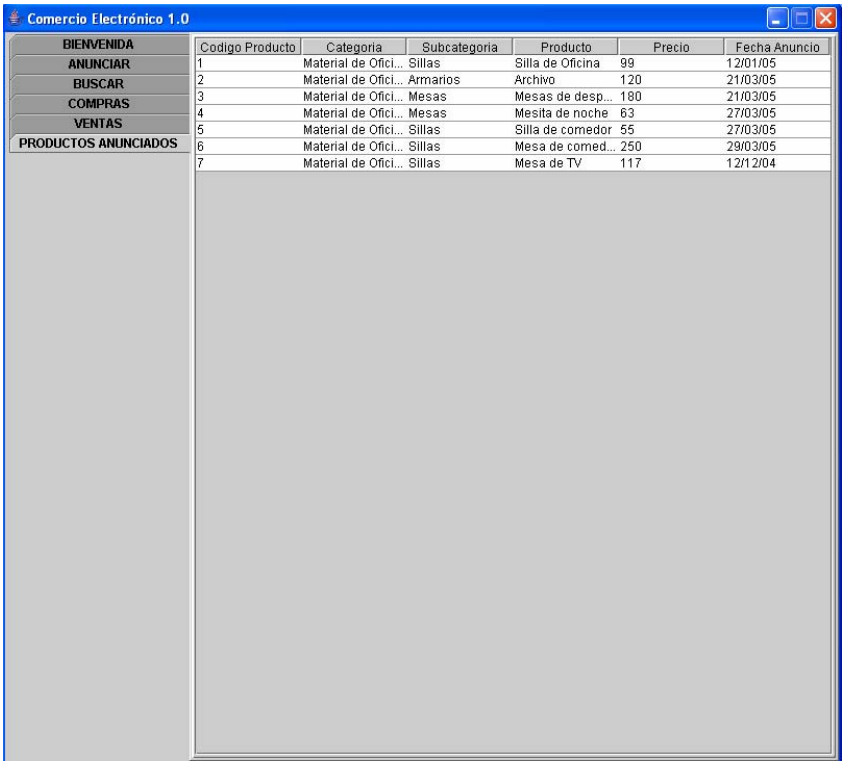
Una vez que se tiene el fichero XML con la descripción del producto y los datos del vendedor, se procede a anunciar un servicio en la red JXTA cuyo nombre será el nombre del producto, por ello es muy importante que al introducir los datos del producto, que se rellene el campo "Nombre del Producto", ya que este nombre será el que posibilite a otros usuarios encontrar nuestro producto en a red.

A cada producto que anunciemos se le asocia un "Código de Producto", que nos va a permitir distinguir los distintos productos que anunciemos. Este código es automáticamente asignado por la aplicación, empezando por el 1 y de forma ascendente. Será este código el que se use también para asignarle un nombre al fichero de extensión ".xml" que contiene los datos del producto.

Todos los ficheros de los productos anunciados se irán

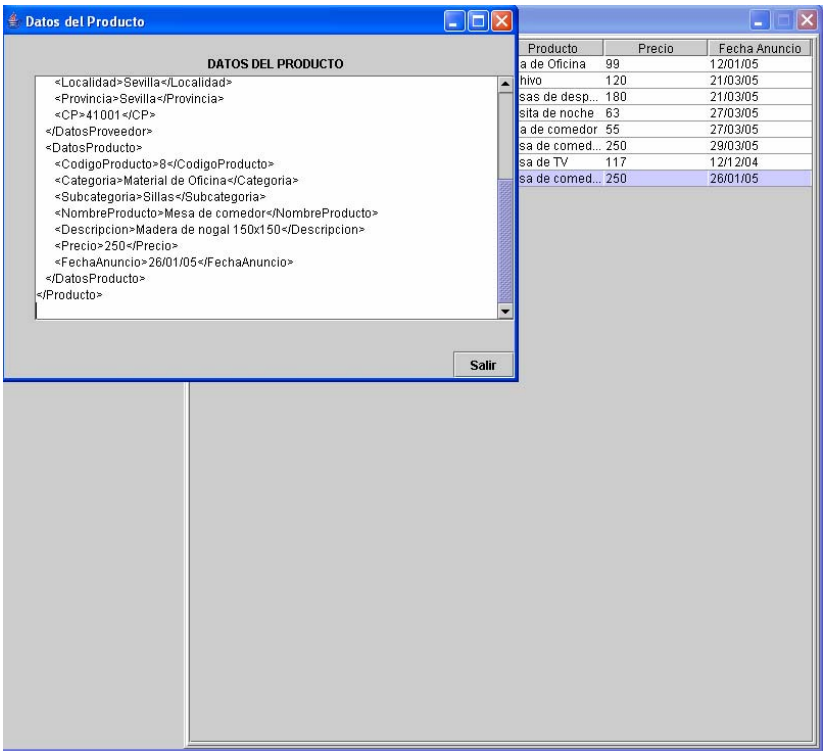
almacenando en el directorio "Anuncios".

El usuario en cualquier momento puede consultar qué productos tiene anunciados pinchando en la pestaña "Productos Anunciados". Al hacerlo se le muestra una tabla con todos los productos que tiene anunciados y que todavía no han sido vendidos.



En la tabla se presenta cada producto en una fila y se muestra los atributos más significativos del mismo: Código del Producto, Categoría, Subcategoría, Nombre del Producto y Fecha del Anuncio.

Para mostrar la información completa del producto, tan solo habrá que pinchar en la fila de la tabla correspondiente y se nos mostrara al completo el archivo XML correspondiente al producto seleccionado.



7.3.2. Buscar un producto.

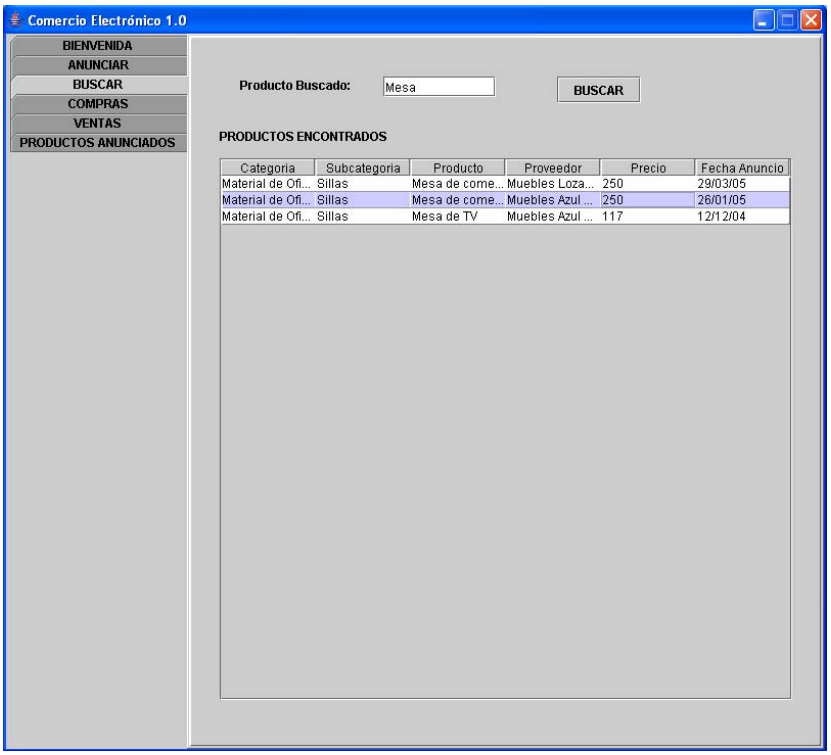
El usuario podrá también en un momento determinado realizar la búsqueda de un determinado producto, para ello tan solo debe escribir en el cuadro destinado para ello, el nombre del producto a buscar.

En el momento en que se pulsa el botón "Buscar", comienza la búsqueda de anuncios de servicios con ese nombre, en la red JXTA dentro del grupo al que nos hemos conectado. En cuanto a la búsqueda decir que para que un servicio se encuentre, la cadena que introducimos debe estar incluida, bien en su totalidad o bien en parte, en la cadena del nombre del servicio

De esta forma todos los usuarios de la red que hayan anunciado servicios con ese nombre deben responder al usuario solicitante enviándole el archivo XML con los datos del producto. Se establece de esta forma una tubería bidireccional JXTA, por cada uno de los productos encontrados, entre lo que llamaremos servidor del producto y cliente del producto.

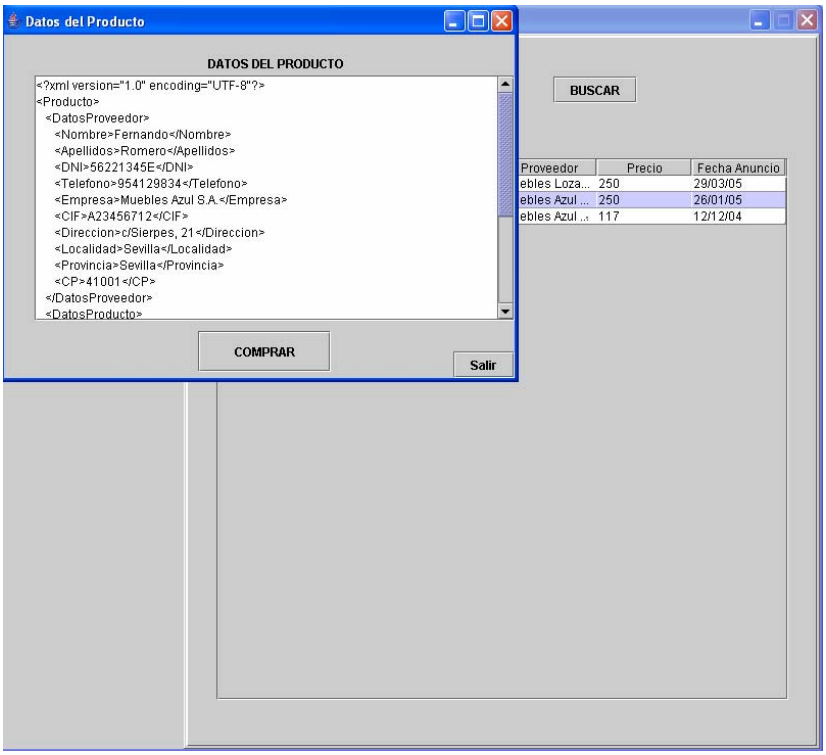
A medida que se van recibiendo los ficheros de productos éstos se van almacenando en el directorio llamado "Encontrados", con extensión ".xml", y nombre numérico, empezando desde el 1 y en orden ascendente. A su vez los productos encontrados se van reflejando en la

tabla de este panel conforme son encontrados.



En esta tabla se muestran al usuario las principales características de los productos encontrados y que se extraen de los fichero XML asociados a cada producto, y son: "Categoría","Subcategoría", "Nombre del Producto", "Nombre del Proveedor", "Precio" y "Fecha del Anuncio".

Si el usuario desea ver la información completa correspondiente a un determinado producto, tan solo debe seleccionar la fila de dicho producto y se le mostrara el archivo XML al completo.

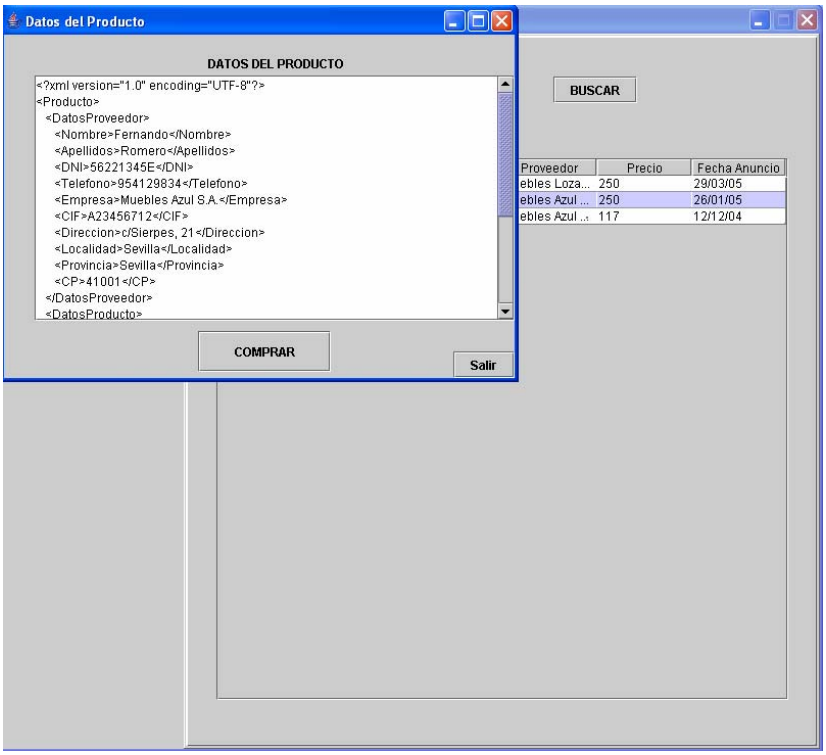


Cada vez que se realiza una nueva búsqueda, esto es, cada vez que se pulsa el botón "Buscar", se eliminarán todos los ficheros almacenados en el directorio "Anunciar" a la vez que se borraran todos los productos que aparecen en la tabla de "Productos Encontrados".

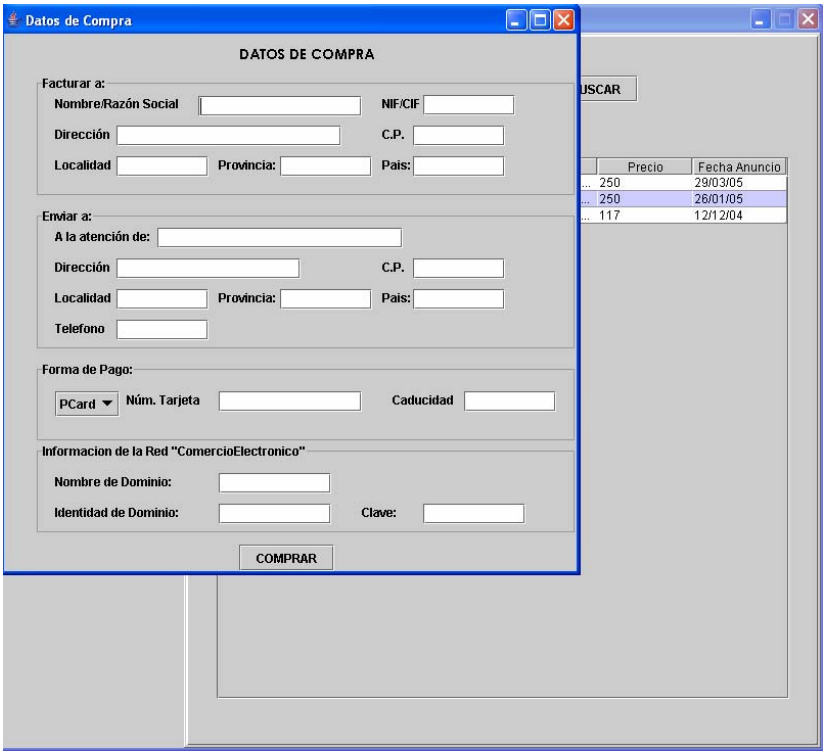
7.3.3. Comprar un producto.

Una vez que al usuario solicitante de un producto le han respondido todos los usuarios de la red que poseen dicho producto, éste puede ir comparando los distintos productos en precio, versión, garantía, plazo de entrega... y finalmente realizar la compra del que más le interese.

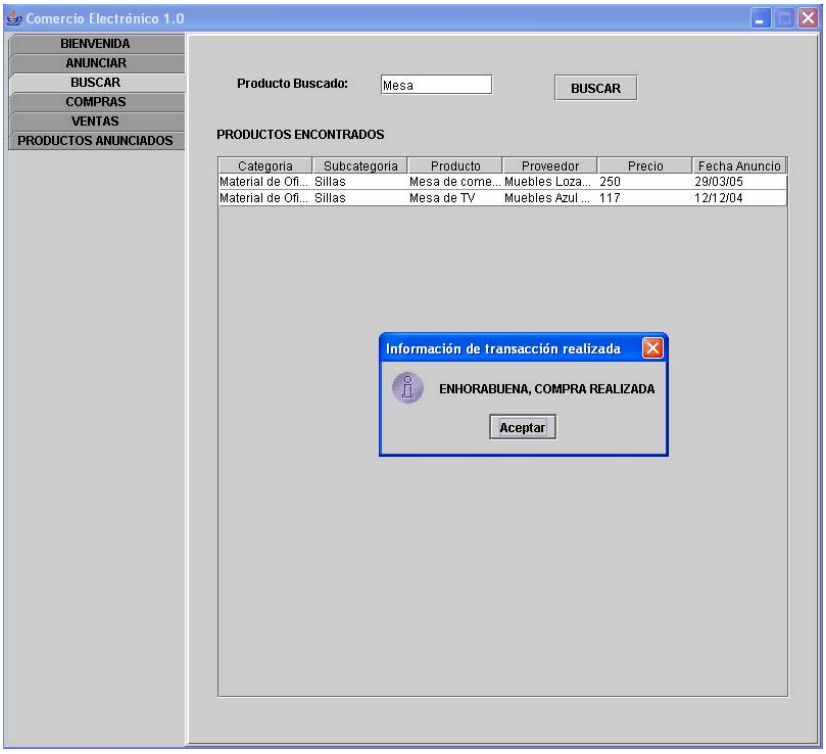
Para ello tan solo deberá seleccionar de la tabla de "Productos Encontrados" el producto que más le interese, se le mostrará entonces una nueva ventana con los datos completos del producto, donde aparecerá el botón "Comprar", además del botón "Salir" como en otras ocasiones.



En el momento de pulsar dicho botón se solicitará al usuario la información necesaria para elaborar los documentos que requiere el estándar cXML, que son dos, por una parte la petición de perfil, ProfileRequest, y por otro la petición de compra, OrderRequest.

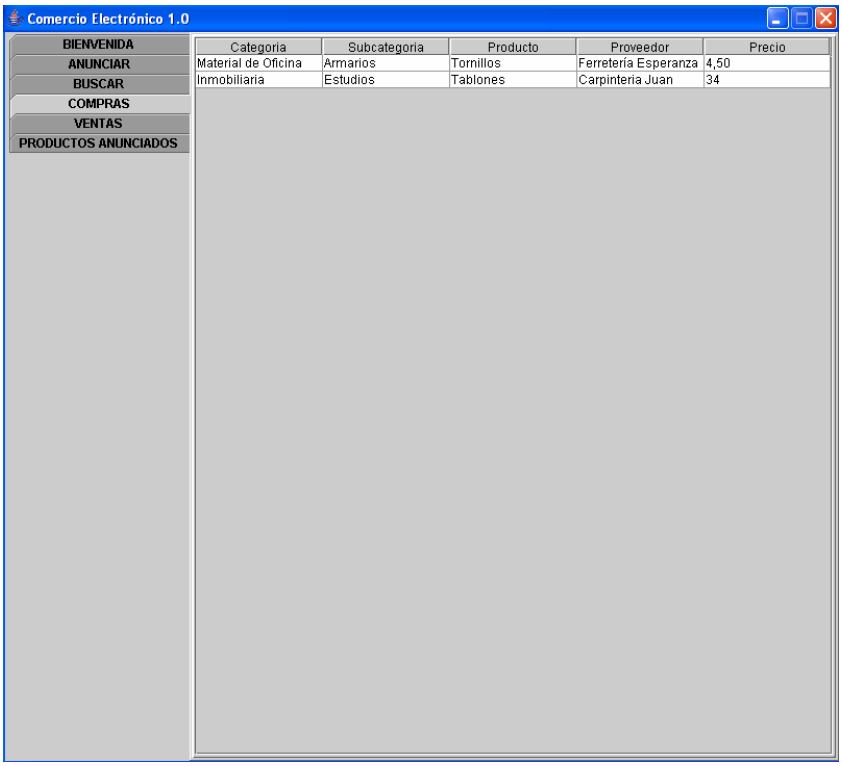


A la hora de pulsar el botón "Comprar" que aparece en este panel, comienzan las transacciones cXML a través de la tubería bidireccional JXTA que se estableció en la búsqueda de productos. En primer lugar y como indica el estándar se envía la petición de perfil, ProfileRequest, hasta que no se reciba por parte del proveedor del producto la respuesta de perfil, ProfileResponse, no se enviará la petición de compra, OrderRequest, a la que el proveedor debe responder con la respuesta a dicha petición, OrderResponse. En este momento aparecerá un panel que nos indicará que la transacción se ha realizado con éxito.



En el momento en que se compra un determinado producto, su fichero XML pasa del directorio "Encontrados" en que se encontraba al directorio "Comprados". La numeración de dicho fichero cambiará pues los productos comprados llevan su propia numeración independiente, comenzando desde el 1 e incrementándose a medida que se compra un nuevo producto.

Así mismo el producto comprado ya no se mostrará en la tabla de "Productos Encontrados", y pasará a mostrarse en la pestaña de "Compras", en la que como en casos anteriores los productos comprados se muestran mediante filas en una tabla.



Por otra parte decir que las peticiones de compra enviadas también se van almacenando en el directorio "PeticionesDeCompraEnviadas", la numeración de dichos ficheros es exactamente la misma que la de los ficheros de productos comprados, es decir, la numeración del fichero de un determinado producto comprado será la misma que la del fichero de su correspondiente petición de compra. Al almacenar las peticiones de compra que enviamos tenemos de alguna forma un justificante de la compra

7.3.4. Vender un producto.

La venta de los productos anunciados es completamente automática. En cualquier momento, cualquiera de los productos que tenemos anunciados puede ser vendido.

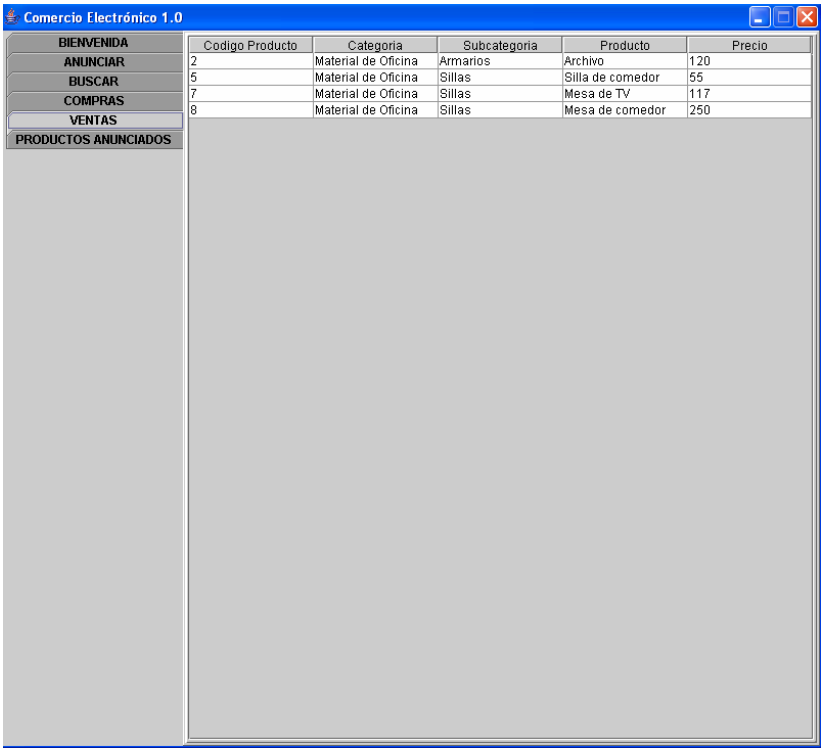
El procedimiento es el siguiente: como ya hemos dicho en el momento que un usuario busca un servicio JXTA cuyo nombre coincide bien en parte o bien totalmente con uno de los productos que tenemos anunciados, se crea una tubería bidireccional entre nuestra aplicación y la aplicación del que realiza la búsqueda.

Será a través de esta tubería por donde recibamos los mensajes cXML para realizar la transacción comercial. En primer lugar recibiremos como ya hemos dicho una petición de perfil, ProfileRequest, a la que

nuestra aplicación responderá automáticamente con una respuesta de perfil, ProfileResponse. Acto seguido debemos esperar una petición de compra, OrderRequest, para poder enviar la respuesta de compra, OrderResponse. La petición de compra recibida la guardaremos en el directorio "PeticionesDeCompraRecibidas", siguiendo la nomenclatura numérica utilizada hasta ahora, de esta forma disponemos de un justificante de la venta.

En este momento el fichero del producto que se encontraba en el directorio "Anuncios", pasarán automáticamente al directorio "Vendidos", conservando la misma numeración. También se procederá a parar el servidor que responde automáticamente a las peticiones de dicho producto y a borrar el anuncio del servicio, de forma que ese producto desaparezca de la red de "Comercio Electrónico P2P".

Por otra parte indicar que al vender un determinado producto, éste desaparece de la tabla de "Productos Anunciados", para pasar a la tabla de la pestaña "Compras", donde al igual que antes se muestran los distintos productos vendidos mediante filas.



7.3.5. Otras funcionalidades.

Conviene decir que cada vez que se inicie el programa deben anunciarse de manera automática todos los productos que se hubieran

anunciado en sesiones anteriores y que todavía no se hayan vendido. Los ficheros XML de estos productos deben encontrarse en el directorio "Anuncios".

Por otra parte y también en ese momento deben leerse los directorios "Vendidos" y "Comprados" de forma que dichos productos se reflejen en las distintas tablas de nuestra aplicación.

Con esto concluimos el apartado dedicado a la explicación de la aplicación desarrollada, en la que se han detallado, por una parte, las clases que la componen y, por otra, las distintas funcionalidades que tiene.

8. MANUAL DE INSTALACIÓN

A continuación se explican los pasos necesarios para instalar y ejecutar la aplicación desarrollada en este proyecto, así como las herramientas necesarias para su buen funcionamiento.

8.1. INSTALACIÓN DE JAVA

En el directorio `software` del CD suministrado en este proyecto viene incluida una versión de un SDK de Java. Es obligatorio el uso de una SDK para esta aplicación. Para instalar, hay que hacer doble clic en el archivo `j2sdk-1_4_2_04-windows-i586-p-iftw.exe` y seguir los pasos. También se puede descargar desde el sitio http://java.sun.com.

Es necesario proporcionar la ruta hacia la máquina virtual de Java instalada. Para ello, hay que incluirla en el *path* del sistema. Para ello debemos editar las variables de entorno del sistema operativo. En caso de usar WindowsXP, esto se consigue pulsando el botón derecho del ratón sobre el icono de "Mi PC" y seleccionar el panel de "Opciones Avanzadas" y pulsar el botón de "Variables de entorno" que aparece en este.

Es entonces cuando se nos despliega un nuevo panel donde se pueden modificar las variables de usuario y las variables del sistema. Entre esas últimas encontramos la variable "Path", que será la que debemos modificar añadiendo, separado por punto y coma, el directorio de comandos de Java, que si no lo hemos modificado durante la instalación será:

C:\j2sdk1.4.2_04\bin

No es necesario configurar el *classpath*, aunque se recomienda. Para que tenga efecto, es necesario reiniciar el ordenador.

8.2. INSTALACIÓN DE "COMERCIO ELECTRONICO 1.0"

Para instalar la aplicación "Comercio Electrónico 1.0" tan sólo hay que copiar todo el directorio del CD "PFC" en cualquier ubicación del disco duro del ordenador. Se puede cambiar el nombre del directorio, para personalizarlo. Aunque es importante no modificar los nombres de los directorios que cuelgan de "PFC", ya que nuestra aplicación intentara acceder a directorios con estos nombres.

Dentro del directorio "PFC" se encuentran los siguientes directorios:

- **"\codigo":** En este directorio se encuentran todas las clases desarrolladas para este proyecto, como los paquetes necesarios de las tecnologías JXTA y JDOM.

- **"\Anuncios":** Directorio donde se almacenarán los ficheros XML de todos los productos que el usuario tiene anunciados y todavía no ha vendido.

- **"\Encontrados":** Directorio donde se almacena los ficheros XML de los productos encontrados durante una búsqueda.

- **"\Vendidos":** Directorio donde se almacenan los productos que el usuario ha vendido.

- **"\Comprados":** Directorio donde se almacenan los ficheros XML de los productos que el usuario ha comprado.

- **"\PeticionesDeCompraEnviadas":** Directorio donde se almacenan los ficheros cXML, "PurchaseRequest" correspondientes a las peticiones de compras enviadas a la hora de comprar un determinado producto encontrado.

- **"\PeticionesDeCompraRecibidas**": Directorio donde se almacenan los ficheros cXML, "PurchaseRequest" correspondientes a las peticiones de compras recibidas al solicitarse al usuario la compra de uno de los productos que tiene anunciados.

8.3. EJECUCIÓN DE "COMERCIO ELECTRONICO 1.0"

Una vez instalada la aplicación en un directorio, se abre una ventana de MSDOS, se accede al contenido de dicho directorio, "PFC", y se teclea:

```
>run
```

Se ejecutará entonces el fichero "run.bat" que se encuentra en el

directorio "PFC" y cuyo contenido es el siguiente:

```
java  -classpath
..;.\codigo\lib\jxta\jxta.jar; .\codigo\lib\jxta\javax.servlet.jar;
.\codigo\lib\jxta\jxtaext.jar; .\codigo\lib\jxta\org.mortbay.jetty.jar;
.\codigo\lib\jxta\log4j.jar; .\codigo\lib\jxta\bcprov-jdk14.jar;
.\codigo\lib\jdom\jdom.jar; .\codigo\lib\jdom\jaxen-core.jar;
.\codigo\lib\jdom\jaxen-jdom.jar; .\codigo\lib\jdom\saxpath.jar;
.\codigo\lib\netbeans\AbsoluteLayout.jar; .\codigo\lib\netbeans\juit-
3.8.1.jar; .\codigo\lib\netbeans\junit-testrunner.jar;
.\codigo\lib\netbeans\jsp-api-2.0.jar; .\codigo\lib\netbeans\servlet-
api-2.4.jar; .\codigo\lib\netbeans\TimerBean.jar;
.\codigo\codigoPFC.jar;  PFC.codigo.Comercio
```

8.3.1. Configuración de JXTA

En este apartado se pretende dar una guía rápida de configuración de JXTA. De todas maneras, se aconseja consultar la documentación de JXTA para una explicación más profunda.

La primera vez que se instancia JXTA, aparece una ventana de configuración, denominada *JXTA Configurator*, en la que se piden diversos parámetros de configuración. Es vital una correcta configuración para el buen funcionamiento de cualquier aplicación basada en JXTA. Las siguientes veces que se instancie JXTA esta pantalla no aparecerá, tampoco se nos volverá a pedir nombre de usuario y contraseña, aunque si por cuestiones de seguridad lo necesitáramos sería posible configurarlo de forma que cada vez que arranquemos la aplicación se nos solicite nombre de usuario y contraseña.

JXTA emplea un directorio, llamado `.jxta`, para almacenar la configuración y los anuncios descubiertos. Este directorio no debe ser modificado, en principio, bajo ningún concepto. Para más información, consultar la documentación de JXTA.

Configuración básica

En la mayoría de los casos, esta configuración es suficiente. Si se desea tener acceso al exterior de la subred local, es necesario realizar la configuración avanzada.

Cuando se inicie una aplicación basada en JXTA, como las de este proyecto, aparecerá el configurador de JXTA:

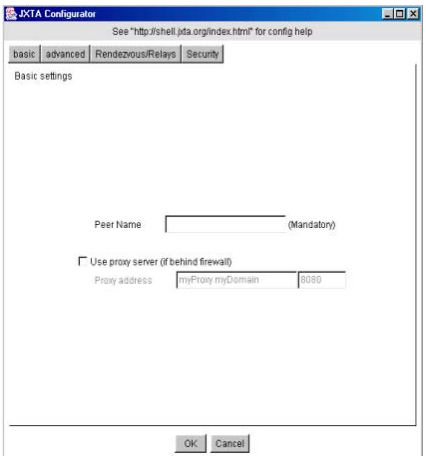


Figura 27: pestaña "Basic"

En la Figura 27: hay que poner un nombre al peer, con el que se identificará en la red JXTA. Si el ordenador está detrás de un servidor *proxy*, hay que especificar la dirección.

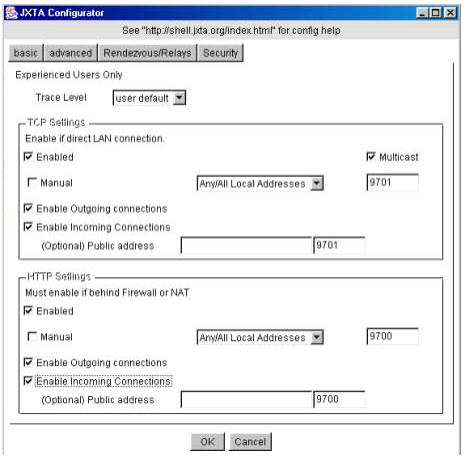


Figura 28: pestaña "Advanced"

En la Figura 28: hay que activar la casilla "*Enable Incoming Connections*" del grupo *"HTTP Settings"* (esto no es necesario la mayoría de las veces). Si hay dos peers ejecutándose en un mismo ordenador, es preciso cambiar los puertos, por ejemplo, poner 9777 y 9778.

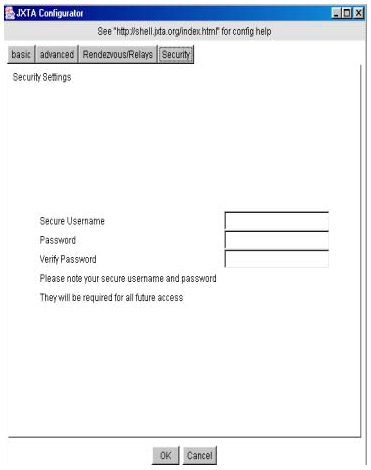


Figura 29: pestaña "Security"

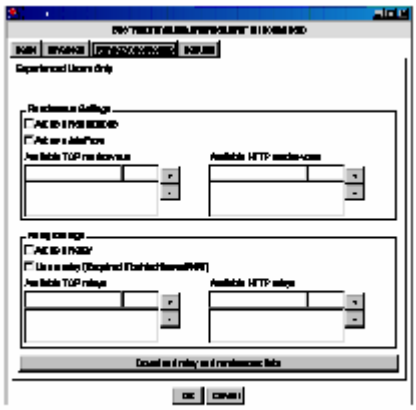
En la Figura 29: hay que suministrar un nombre y clave. Tal y como se ha configurado para este proyecto no se le pedirá al usuario otras veces que se instancie JXTA. No define ningún mecanismo de seguridad realmente.

Esta ventana sólo aparecerá la primera vez que se instancie una aplicación en un directorio.

Para borrar la configuración, se debe borrar el directorio `.jxta` creado automáticamente en el directorio donde se encuentra la aplicación.

Configuración avanzada

Consiste en los mismos pasos que la básica, pero hay que configurar la pestaña "Rendezvous/Relay" del configurador, para permitir el acceso al exterior de la red local:



Sólo hay que presionar el botón "*Download relay and rendezvous list*" y cargar las listas de las direcciones que aparecerán.

De todas maneras, la configuración de JXTA es complicada y depende mucho del caso particular de cada conexión. Lo más aconsejable es consultar la documentación al respecto en www.jxta.org.

9. PLANIFICACIÓN TEMPORAL DEL PROYECTO

El proyecto se ha divido en siete tareas principales:

- **Tarea de documentación.** En ella, se han ido adquiriendo los documentos, en plan general, tanto en formato papel como electrónico, de las distintas tecnologías posibles para el desarrollo del proyecto así como las herramientas necesarias para la realización de la aplicación.

- **Tarea de estudio de la tecnología JXTA.** Durante esta etapa se entró en el estudio específico de esta tecnología, por una parte completando la documentación obtenida en a fase anterior y aprendiendo el funcionamiento de la misma.

- **Tarea de estudio de la tecnología XML y cXML.** En esta fase se entró al estudio detallado de ambos lenguajes, completando la documentación obtenida en la primera fase.

- **Tarea de programación de la aplicación.** En esta tarea, se ha llevado a cabo el diseño, implementación y prueba de la aplicación realizada.

- **Tarea de pruebas finales.** Período en el que se han hecho pruebas al sistema completo y donde se ha comprobado su correcto funcionamiento.

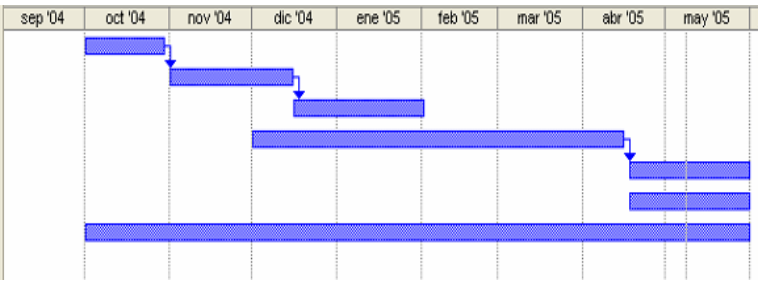
- **Tarea de redacción de la memoria del proyecto fin de carrera.**

- **Tarea de seguimiento del proyecto fin de carrera.** Periódicamente, se ha informado a los profesores tutores del estado de ejecución de las otras tareas. Así, ha podido seguir la evolución del proyecto y establecer las medidas correctivas que ha estimado oportunas. La información necesaria para el seguimiento del proyecto, se ha hecho llegar mediante correos electrónicos y reuniones en su despacho.

A continuación se muestra la planificación del proyecto mediante

un análisis de tareas y un diagrama de Gantt:

	Nombre de tarea	Comienzo	Fin
1	Documentación previa	vie 01/10/04	vie 29/10/04
2	Estudio de la tecnología JXTA	lun 01/11/04	mié 15/12/04
3	Estudio de la tecnología XML y cXML	jue 16/12/04	mar 01/02/05
4	Programación de la aplicación	mié 01/12/04	vie 15/04/05
5	Pruebas finales	lun 18/04/05	mar 31/05/05
6	Redacción de la memoria	lun 18/04/05	mar 31/05/05
7	Seguimiento del proyecto	vie 01/10/04	mar 31/05/05



10. PRESUPUESTOS

En la elaboración del presupuesto de este proyecto se han tenido en cuenta los aspectos que se detallan a continuación:

- Ha sido desarrollado por una sola persona cuyo nivel profesional puede corresponder a ingeniero júnior.

- La ejecución del proyecto ha comprendido el período de tiempo que va de Octubre de 2004 a Mayo de 2005 (ambos inclusive).

Se ha desglosado el presupuesto de la siguiente manera:

- Coste de recursos humanos.

- Coste de material, subdividido a su vez en:

  - Coste del software.

  - Coste del hardware.

  - Coste de consumibles.

10.1. COSTE DE RECURSOS HUMANOS

El coste presupuestario de personal se ha calculado de la siguiente forma. Si se supone que se ha desempeñado una jornada de trabajo básica de 40 horas semanales y que se puede establecer un sueldo base de unos 1.500 € mensuales para un ingeniero júnior, el coste de recursos humanos es:

8 meses (Octubre´04 a Mayo´05) x 1500 €……..12.000 €

Subtotal	**12.000 €**

10.2. COSTE DE MATERIAL

10.2.1. Coste del software.

El software usado es en su mayor parte de libre distribución, no obstante, el sistema operativo y paquete ofimático son propietarios puesto que son ampliamente usados, y estaban instalados en el ordenador usado en este proyecto. Sin embargo, se podría haber elegido cualquier opción de software libre, con el consiguiente ahorro.

Microsoft Windows XP Home Edition……117 €
Microsoft Office XP Home Edition……455 €

Subtotal	**572 €**

10.2.2. Coste del hardware.

El hardware empleado para desarrollar todo el proyecto ha sido un ordenador personal, una impresora y una memoria USB (para transporte de datos voluminosos). Por lo tanto, el coste de hardware es el siguiente:

1 PC……900 €
1 Impresora……150 €
1 Memoria USB……55 €

Subtotal	**1105 €**

10.2.3. Coste de consumibles.

Como consumible se entiende todo aquel recurso que es susceptible de ser consumido, como papel, electricidad, etc.

Electricidad consumida……..60 €

Material de oficina……60 €

8 Meses de conexión a Internet……..240 €

Subtotal	**360 €**

10.3. RESUMEN DEL PRESUPUESTO

A continuación se resume el presupuesto general del proyecto, consistente en la suma de los presupuestos anteriormente citados:

Coste de Recursos Humanos……..12.000 €

Coste de Material:

Coste de Software…… 572 €

Coste de Hardware……..1.105 €

Coste de Consumibles…….. 360 €

TOTAL	**14.037 €**

11. LINEAS FUTURAS.

Cabe decir que el objetivo fundamental de este proyecto ha sido desarrollar un prototipo para comercio electrónico, en el que el usuario sea capaz de buscar productos, comprarlos, anunciarlos y venderlos. Por tanto no se ha entrado en el estudio detallado de ciertos aspectos necesarios si quisiéramos darle un uso real a la aplicación.

Entre estos aspectos cabe destacar:

11.1. CENTRALIZACIÓN

Como se puede comprobar la aplicación desarrollada en este proyecto funciona en una red totalmente descentralizada, es decir, una red entre iguales, ningún nodo tiene mayor potestad que otros. Quizá sería interesante orientar la aplicación de forma que existiera un organismo centralizado que garantizara la autenticidad de las transacciones.

Las funciones de esta entidad central serían:

- Llevar un control de todos los usuarios que se incorporan y abandonan la red "Comercio Electrónico", de forma que todo usuario antes de poder realizar transacciones comerciales en la red debería registrarse en dicho organismo y obtener por parte de este algún tipo de código que le identifique de forma única en la red.

- Gestionar las categorías y subcategorías con que se clasifican los productos, de forma que éstas fueran dinámicas y fueran actualizándose en función de las necesidades de los usuarios. De esta forma esta entidad central se encargaría de enviar al usuario el fichero de categorías actualizado cada vez que este se conecte a la red, a su vez el usuario tendría la capacidad de interactuar con la entidad central para añadir una nueva categoría si ésta no estuviera contenida en dicho fichero.

- Llevar un registro de las transacciones comerciales para garantizar el no repudio y la autenticidad de éstas.

11.2. SEGURIDAD

Es uno de los aspectos fundamentales de todos los sistemas actuales, y en este proyecto cobra vital importancia, ya que se realizan transacciones económicas. El estudio e implantación de políticas de seguridad supone una investigación larga, costosa y compleja, sobre todo por la novedad de las tecnologías empleadas en este proyecto.

La investigación e implantación de mecanismos de seguridad afecta a todas las partes del proyecto, y deben colaborar para, al menos, minimizar la probabilidad de riesgo y evitar daños graves.

A continuación se exponen varias líneas basadas exclusivamente en la seguridad:

- **Seguridad en las comunicaciones**: se debe investigar e implantar mecanismos seguros de transmisión de datos a través de la red *peer-to-peer*. Para ello se debe estudiar el estado actual de la seguridad en el proyecto JXTA (existe un grupo de trabajo dedicado a ello), e implantar las acciones propuestas. Si cabe, sería conveniente la definición de otras políticas, como:

    - Cifrado de datos usando algoritmos de clave pública.

    - Creación de tuberías de comunicación inherentemente seguras.

    - Autenticación a nivel de enlace extremo a extremo en la red *peer-to-peer*.

  La implantación de estas medidas no excluye su implantación en las capas superiores de comercio electrónico.

- **Seguridad en los datos**: se deben estudiar los mecanismos de encriptación, autenticación y autorización implantables en documentos cXML. Esto incluye tecnologías de firmas digitales, encriptado de XML, etc. A pesar de que todos estos mecanismos se pueden y deben también implantar en el nivel de comunicaciones, *peer-to-peer* en este caso,

también debe implantarse en este nivel para mayor redundancia, y así aprovechar las ventajas que ofrece XML en materia de seguridad.

Como ejemplo, se puede citar el uso de 2 tecnologías recientemente propuestas por el W3C para la seguridad en XML: *XML Encryption* y *XML Signature*.

- **Seguridad en la red *peer-to-peer***: este aspecto es más complicado, puesto que exige la implantación de mecanismos más complejos y experimentales. Consiste en el desarrollo de acciones dentro de la red *peer-to-peer*, JXTA en este caso, para promover un entorno seguro y de confianza. La línea básica es la creación, gestión y monitorización de comunidades *peer-to-peer* orientadas al comercio electrónico que implementen mecanismos de seguridad como:

  - Detección y corrección antivirus distribuida.

  - Políticas de seguridad de acceso y ejecución.

  - Políticas de confianza, mediante sistemas de membresía, popularidad, perfiles, etc. También son válidas las medidas con participantes externos de confianza.

Es evidente que el tema de la seguridad es muy amplio y está constantemente en renovación, puesto que nuevas amenazas aparecen continuamente. Por ello, se hace necesaria una continua investigación e implantación de las técnicas más actuales.

11.3. MAYOR ESPECIALIZACIÓN

Este proyecto desarrolla una aplicación de propósito general, es decir, sería válida para cualquier usuario que quisiera vender un determinado producto. Pensando en la heterogeneidad y en el elevado número de usuarios que harían uso de esta herramienta, sería útil que hubiera distintas redes especializadas en función por una parte del tipo de producto y por otra parte del tipo de transacción comercial (B2B, B2C, C2C…)

Hay que tener en cuenta que las formas de pago, condiciones de compra, los precios, garantías… dependen mucho del tipo de transacción comercial, esto hace que adquiera sentido la teoría de hacer redes especializadas.

11.4. MEJORAS TECNOLÓGICAS.

Al hablar de líneas futuras de desarrollo no podemos olvidarnos de las cuestiones tecnológicas. La gran cantidad de tecnologías novedosas empleadas en la realización de este proyecto, y la temática en sí de la convergencia de *peer-to-peer* con comercio electrónico, posibilita enormemente su ampliación. Las ideas básicas que se pretenden llevar a cabo para extender este proyecto son:

- **adaptarlo** a las nuevas versiones de los estándares utilizados.

- **mejorar** la implantación de las tecnologías usadas, como por ejemplo:

  - refinar el código para aumentar la eficiencia y compatibilidad.

  - añadir nuevas funcionalidades descritas en los estándares y no usadas en este proyecto.

  - desarrollar aplicaciones más potentes y con mayor número de capacidades.

- **proporcionar compatibilidad** entre las tecnologías usadas y otras que estén ampliamente aceptadas.

- **migrar hacia nuevas tecnologías**, que se perfilen como punteras y de mayor proyección hacia el futuro.

En este sentido tiene especial importancia la investigación del estándar de comercio electrónico **ebXML** y la posible adaptación de esta aplicación a dicho estándar. Está desarrollado por la UN/CEFACT, por lo que goza de mucha popularidad y apoyo por parte de numerosos consorcios, empresas y organismos públicos de todas las naciones. El objetivo de ebXML es la creación de un mercado

electrónico único y global donde cualquier empresa u organización, sin importar la localización geográfica, pueda establecer relaciones económicas con otros, mediante el intercambio de mensajes XML.

ebXML no sólo define el "lenguaje" (como hace cXML), sino todo un modelo de negocio. Las especificaciones son complicadas de entender y, dada su relativa cercana fecha de aparición, aún no abundan herramientas y documentación que faciliten el desarrollo de aplicaciones.

Sin embargo, es previsible que ebXML cobre mucha importancia en el futuro inmediato o, al menos, sea el referente para otros estándares más implantados. Por ello, se hace imprescindible una investigación sobre ebXML y su posible aplicación a este proyecto. En caso de ser apropiado, sería conveniente la implantación de ebXML sobre *peer-to-peer*, pudiendo sustituirse por cXML o dar soporte de compatibilidad entre ambos sistemas.

12. CONCLUSIONES

Es una realidad que el comercio electrónico está creciendo de una forma exponencial, prueba de ello son los datos que se manifiestan en los informes a los que se hace alusión en esta memoria.

Con este proyecto se consigue cambiar la filosofía actual del comercio electrónico B2C, basada en portales del tipo "carro de la compra" donde el usuario adquiere el producto que le interesa.

La competencia en el comercio electrónico radica principalmente en la capacidad que tiene el proveedor de llegar al cliente. Por muy buenos precios y prestaciones que ofrezca una determinada empresa si no es capaz de llegar a sus clientes potenciales no sirve de nada.

El modelo que aquí se plantea favorece la libre competencia, pues los productos competirán en igualdad de condiciones, por sus prestaciones y no por la capacidad de colocarlos en un determinado portal con más audiencia o prestigio.

En cuanto al comercio electrónico business-to-business, que como hemos dicho es el que más volumen mueve, también plantea una revolución, ya que supone un ahorro de tiempo y dinero en las transacciones comerciales.

Quizá sería necesario hacer pruebas con un gran número de usuarios a nivel mundial para comprobar la escalabilidad de la aplicación y el potencial del motor de búsqueda. Es evidente que para el desarrollo de este proyecto no se disponía de los recursos necesarios para ello.

En cuanto a las tecnologías utilizadas, decir que son todos estándares ampliamente aceptados y todos bastante novedosos a su vez, lo que por otra parte hace que no hayan alcanzado su madurez y haga falta esperar algunos años para que sea una verdadera alternativa. Pero se comprueba que a pequeña escala que es donde se ha probado la aplicación los resultados obtenidos son satisfactorios.

En cuanto al modelo peer-to-peer introduce el concepto de igualdad en las redes. No tiene porqué haber papeles preponderantes,

sino que la red puede estar constituida por integrantes iguales, idénticos en derechos y deberes. Y si se prefiere establecer jerarquías, la arquitectura p2p es lo suficientemente flexible como para permitirlo sin traicionar sus principios. Se abandona de esta forma el clásico modelo cliente-servidor como ya está pasando en determinadas áreas como puede ser la música.

13. BIBLIOGRAFÍA

Se citan a continuación las principales fuentes bibliográficas, clasificadas temáticamente, que se han consultado a la hora de desarrollar este proyecto.

13.1. ARQUITECTURA "PEER TO PEER":

- "Looking Up Data in P2P Systems", Hari Balakrishnan, M. Franks Kaashoek. Communications of the ACM, Feb 2003.

- "An Introduction to Peer-To-Peer Computing" David Barkai, Intel Corporation.

  http://www.intel.com/update/departments/initech/it02012.pdf

- Wikipedia

  http://es.wikipedia.org/wiki/P2P

- "Internet y Comercio Electrónico", Julián Briz e Isidro Laso, 2ª Ed. Ediciones Mundi-Prensa

- "P2P en la empresa"

  http://www.terra.es/tecnologia/articulo/html/tec8002.htm

- "P2P Networks"

  http://ntrg.cs.tcd.ie/undergrad/4ba2.02-03/Intro.html

- "An introduction to peer-to-peer networks" Vinod Muthusamy

  http://www.eecg.toronto.edu/~jacobsen/mie456/slides/p2p-mie.pdf

- "Gnutella: Distributed System for Information Storage and

Searching Model Description", Fernando R.A. Bordignon, Gabriel H. Tolosa. Universidad Nacional de Lujan.

- "Freenet: A Distributed Anonymus Information Storage and Retrieval System", Ian Clarke, Oscar Sandberg, Brandon Wiley, Theodore W. Hong.

- "Edutella: A P2P Networking Infraestructure Based on RDF", Wolfgang Nejdl, Boris Wolf, Changtao Qu.

- "Estudio y Definición de Protocolos Compañero a Compañero para soportar Servicios Distribuidos sobre Internet", Gabriel Tolosa, Fernando Bordignon. Universidad Nacional de Lujan, Sep 2003.

- "Redes de contenido: un panorama de sus características y principales aplicaciones", Pablo Rodríguez Bocca, Héctor Cancela Bosi. Universidad de la República de Montevideo, Uruguay.

- "Sistema de comercio electrónico sobre plataforma peer-to-peer", Josef Ignacio Hötz Ordoño, Universidad de Sevilla, May 2004.

13.2. COMERCIO ELECTRÓNICO

- "Informe sobre el comercio electrónico en España a través de entidades de medios de pago (4º Trimestre 2004)", Comisión del Mercado de las Telecomunicaciones.

- "Estudio sobre Comercio Electrónico B2B 2004", Asociación Española de Comercio Electrónico (AECE), May 2004.

- "La Sociedad de la Información en España 2004", Telefónica, Dic 2004.

- "Navegantes en la Red, 7ª encuesta AIMC a usuarios de Internet" , Asociación para la Investigación de Medios de Comunicación, Feb 2005.

- www.e-global.es Portal sobre comercio electrónico y e-business.

- "Internet y Comercio Electrónico", Julián Briz e Isidro Laso. 2ª Ed. Ediciones Mundi-Prensa

- "Designing Systems for Internet Commerce", G. Winfield Treese y Lawrence C.Stewart, Addison Wesley

- "e-Commerce", David Whiteley, Mc Graw Hill

- "eBusiness Essentials", Mark Norris, Steve West y Kevin Gaughan, John Wiley & Sons

- http://www.it.uc3m.es/~xml/enlaces.html (página de enlaces)

13.3. JAVA

- "Manual de Java", Jorge Sánchez. 2004.

- http://www.programacion.com/java/tutoriales/ Página con enlaces a distintos manuales de Java.

- http://java.sun.com/ Página oficial del lenguaje Java.

13.4. JXTA

- "Project JXTA: A Loosely-Consistent DHT Rendezvous Walker", Bernard Traversat, Mohamed Abdelaziz, Eric Pouyoul. Sun Microsystems, Inc.

- "JXTA Search: Distributed Search for Distributed Networks", Steve Waterhouse, Sun Microsystems, May 2001.

- "Overview of JXTA", Sam Publishing

  http://www.developer.com/java/other/article.php/10936_1450221_1

- www.jxta.org Página oficial del proyecto JXTA

- "JXTA v2.0 Protocols Specification"

- "JXTA v2.3.1: Java Programmer´s Guide"

- "JXTA 2: A high-performance, massively scalable P2P network", Sing Li.

  http://www-106.ibm.com/developerworks/java/library/j-jxta2/

13.5. XML

13.5.1. XML, XSL y DOM:

- "Tutorial de XML", Miloslav Nic.

  http://www.zvon.org/xxl/XMLTutorial/General_spa/book.html

- "Manual sobre XML", Joaquín Bravo Montero

  http://www.programacion.net/html/xml/principal.htm

- "Tutorial de XML", Ángel Barbero Paniagua

  http://www.dat.etsit.upm.es/~abarbero/curso/xml/xmltutorial.html

- "XML", Daniel M. Germán

  http://csgrs6k1.uwaterloo.ca/~dmg/tutorial/xml/

- "Introducción a XML"

  http://www.desarrolloweb.com/manuales/18/

- "Apuntes de XML", Diego Arranz Hernando

  http://www.programacion.net/tutorial/apuntesxml/4/

- "The Java Web Services Tutorial"

  http://java.sun.com/webservices/docs/1.2/tutorial/doc/index.html

- "Generación de páginas Web usando XSLT y XML", J. J. Merelo

  http://geneura.ugr.es/~jmerelo/XSLT/

- "Introducción a XML", Guillermo Ontañón

  http://weezer.dragon-lance.net/articulos/xmlintro.html

- "Using XSL-FO to create printable documents", Rodolfo M. Raya

  http://www-106.ibm.com/developerworks/xml/library/x-xslfo/

- Web de ejemplos de uso de XPath

  http://www.zvon.org/xxl/XPathTutorial/Output/example1.html

- Documentación de http://www.xml.com

- Recomendaciones sobre DOM publicadas por W3C en http://www.w3.org/

- "Discover key features of DOM Level 3 Core", Arnaud Le Hors, Elena Litani

- http://www-106.ibm.com/developerworks/xml/library/x-keydom.html

13.5.2. JDOM

- Documentación de http://www.jdom.org

13.5.3. cXML

- Guías del usuario de cXML, versiones 1.1 (en español) y 1.2.011 (en inglés).

- http://www.cxml.org

14. ANEXO: CÓDIGO FUENTE DE LA APLICACIÓN.

A continuación se anexa el código fuente de las clases desarrolladas en este Proyecto Fin de Carrera. Dichas clases han sido organizadas en los siguientes paquetes:

- **PFC.codigo.acciones:** En este paquete se incluye las clases que se corresponden con las distintas acciones que se pueden hacer en el programa

- **PFC.codigo.xml:** En este paquete se incluyen todas las clases relacionadas con el manejo de documentos XML

- **PFC.codigo.jxta:** Paquete que incluye todas las clases relacionadas con la tecnología JXTA.

- **PFC.codigo.tablas:** Se incluyen en este paquete todas las clases relacionadas con el manejo de las tablas de las que consta la aplicación

- **PFC.codigo.utiles:** Paquete con una serie de clases con utilidades en el manejo de ficheros, directorios, números aleatorios…

- **PFC.codigo:** En este paquete solo se define la clase principal del programa, es decir, la clase Comercio.

1.1. PAQUETE PFC.CODIGO.ACCIONES

1.1.1. Clase Anunciar

```
//
//  Fichero: Anunciar.java
//  Autor: Trinitario Meseguer Calero
//  Fecha: 30/11/04
//  Descripción: Esta clase se instancia cada vez que se pulsa el botón "Anunciar" del
//       panel  con el mismo nombre. Las  funciones de esta clase son: anunciar un
//       determinado producto, para lo cual se llama a la clase ServidorServiciosJXTA,
//       y anunciar los productos que se anunciaron en sesiones anteriores y que
//       todavía no han sido vendidos.
//
```

```
package PFC.codigo.acciones;

import PFC.codigo.jxta.*;
import PFC.codigo.tablas.*;
import PFC.codigo.utiles.*;
import PFC.codigo.xml.*;
import PFC.codigo.*;

import java.io.*;
import javax.swing.*;
import javax.swing.table.*;
import java.util.*;


public class Anunciar {

   ProductoXML producto;
   String nombreFichero;
   PlataformaJXTA plataforma;
   ModeloTablaAnunciados modeloAnunciados;
   ModeloTablaVendidos modeloVendidos;
   LinkedList listaAnuncios;


//   Constructor de la clase Anunciar, inicializa ciertas variables locales
//   a partir de los argumentos que se le pasan.
   public Anunciar(ProductoXML producto, String nombreFichero, PlataformaJXTA
plataforma,ModeloTablaAnunciados modeloAnunciados, ModeloTablaVendidos
modeloVendidos, LinkedList listaAnuncios) {

      this.producto=producto;
      this.nombreFichero=nombreFichero;
      this.plataforma=plataforma;
      this.modeloAnunciados=modeloAnunciados;
      this.modeloVendidos=modeloVendidos;
      this.listaAnuncios=listaAnuncios;

   }

//   Método utilizado para anunciar un nuevo producto en la red JXTA, para ello
//   se instancia la clase ServidorServiciosJXTA y se le pasan los
//   argumentos necesarios.
   public void AnunciarProducto(){

       ServidorServiciosJXTA servicioJXTA=new
ServidorServiciosJXTA(plataforma,nombreFichero,
modeloAnunciados,modeloVendidos);
      servicioJXTA.PublicaServicio(producto);
      servicioJXTA.EnlazaAnuncio(listaAnuncios);
      servicioJXTA.EsperaCliente();
```

```
   }

//   Método llamado cada vez que se ejecuta la aplicación,se encarga de
//   anunciar los productos que hay en el directorio "Anuncios" y que son
//   los productos anunciados en sesiones anteriores.
   public void AnunciarProductosAntiguos(){
      String rutaDirectorio="./Anuncios";
       File directorio=new File(rutaDirectorio);
       ProductoXML producto;
       Anunciar anuncio;
      if(directorio.isDirectory()){

         File[] listaFicheros=directorio.listFiles();


         for(int i=0;i<listaFicheros.length;i++){
            try{
               producto=new ProductoXML(listaFicheros[i]);
               anuncio=new
Anunciar(producto,listaFicheros[i].getName(),plataforma,modeloAnunciados,modeloV
endidos,listaAnuncios);
               anuncio.AnunciarProducto();

            }
            catch(Exception e){
               System.out.println("Error al leer el fichero de producto");
            }
         }
      }
      else {
         System.out.println("La ruta especificada no es un directorio");
      }


   }

//   Cada vez que se pulsa el botón "Anunciar" del panel con el mismo nombre
//   se llama a este método que es el responsable de que se borre la
//   información del producto introducida por el usuario en dicho panel. La
//   información relativa al vendedor y a la red "Comercio Electrónico" no se
//   borra pues se considera que pocas veces variará y por  tanto se considera
//   más cómodo que el usuario lo borre manualmente cuando quiera modificarlo,
//   a que lo tenga que rellenar cada vez que vaya a anunciar un nuevo
//   producto.
   public void BorraPanelAnunciar(Comercio claseComercio){


      claseComercio.CampoPrecio.setText("");
```

```
        claseComercio.CampoNombreProducto.setText("");
        claseComercio.CampoPrecio.setText("");
        claseComercio.TextoDescripcion.setText("");
        claseComercio.CampoFechaAnuncio.setText("");

        claseComercio.codigoProducto++;


claseComercio.CampoCodProducto.setText(String.valueOf(claseComercio.codigoProd
ucto));


    }

}
```

1.1.2. Clase Buscar

```
//
//    Fichero: Buscar.java
//    Autor: Trinitario Meseguer Calero
//    Fecha: 27/01/05
//    Descripción: Esta clase se instancia cada vez que se pulsa el botón
//        "Buscar" del panel con el mismo nombre. Se encarga de borrar la
//        tabla de productos encontrados en búsquedas anteriores, también
//        borra  los ficheros del directorio "Encontrados", que se
//        corresponde con los productos encontrados en búsquedas anteriores, y
//        llama a la clase ClienteServiciosJXTA que será la clase
//        encargada de buscar dicho servicio en la red JXTA.
//


package PFC.codigo.acciones;

import PFC.codigo.jxta.*;
import PFC.codigo.tablas.*;
import PFC.codigo.utiles.*;
import PFC.codigo.xml.*;


public class Buscar{
    public ClienteServiciosJXTA cliente;
    /** Creates a new instance of Buscar */
    public Buscar() {
    }

//    Método usado cada vez que se desea buscar un determinado producto en la
//    red JXTA, para ello se borra el contenido del directorio "Encontrados",
//    así como la tabla de productos encontrados con los resultados de
//    búsquedas anteriores y se instancia un objeto de la clase
//    ClienteServiciosJXTA al que se le pasan los argumentos necesarios.
```

```
   public void BuscarProducto(String palabraBusqueda, PlataformaJXTA plataforma,
ModeloTablaEncontrados modeloEncontrados){

      modeloEncontrados.BorraTabla();
      Fichero.BorraContenidoDirectorio("./Encontrados");
      cliente=new ClienteServiciosJXTA(plataforma, modeloEncontrados);
      cliente.BuscarServicio(palabraBusqueda);

   }

}
```

1.1.3. Clase Comprar

```
//
//   Fichero: Comprar.java
//   Autor: Trinitario Meseguer Calero
//   Fecha: 07/03/05
//   Descripción: Las principales funciones de esta clase son:
//      1. Rellenar y enviar la Petición de Perfil propia del protocolo
//      cXML a través de la conexión bidireccional.
//      2. Enviar la petición de Compra al peer que está anunciando
//      el producto a través de la tubería establecida con este.
//      3. Guardar el archivo cXML de Petición de Compra en el
//      directorio “PeticionesDeCompraEnviadas”, para de esta forma tener
//      una prueba de la compra realizada.
//      4. Guardar el archivo XML del producto comprado el directorio
//      “Comprados”, para tener almacenados todos los datos del
//      producto comprado.
//


package PFC.codigo.acciones;

import PFC.codigo.jxta.*;
import PFC.codigo.tablas.*;
import PFC.codigo.utiles.*;
import PFC.codigo.xml.*;

import net.jxta.util.*;
import java.io.*;
import javax.swing.*;


public class Comprar {
   PeticionDeCompraCXML peticionCompra;
   ProductoXML producto;
   JxtaBiDiPipe bipipe;

   String rutaDirectorioPeticionCompra="./PeticionesDeCompraEnviadas/";
   String rutaDirectorioProductoComprado="./Comprados/";
```

```
   ModeloTablaComprados modeloComprados;

//  Constructor de la clase Comprar que inicializa ciertas variables locales
//  a partir de los argumentos que se le pasan
   public Comprar(PeticionDeCompraCXML peticionCompra, ProductoXML producto,
TuberiaCliente tuberia, ModeloTablaComprados modeloComprados) {

      this.peticionCompra=peticionCompra;
      this.producto=producto;
      this.bipipe=tuberia.bipipe;
      tuberia.compra=this;
      this.modeloComprados=modeloComprados;
   }

//   Método encargado de rellenar y enviar la petición de perfil del
//   protocolo cXML
   public void enviarPeticionPerfil(){

      PeticionDePerfilCXML peticionPerfil=new PeticionDePerfilCXML();
      peticionPerfil.setDominioCredencialTo(producto.dominio.getText());
      peticionPerfil.setIdentidadTo(producto.identidad.getText());
      System.out.println("Se envia Peticion de Perfil");
      peticionPerfil.enviar(bipipe,"PROFILE_REQUEST");

   }

//   Método encargado de enviar la petción de compra del protocolo cXML
   public void enviarPeticionCompra(){
      peticionCompra.enviar(bipipe,"ORDER_REQUEST");
   }

//   Método responsable de traspasar el fichero de un producto encontrado al
//   directorio de productos comprados cuando la venta se ha realizado, así como
//   de insertar dicho producto en la tabla del panel de productos comprados.
   public void compraRealizada(){

      File directorioComprados=new File(rutaDirectorioProductoComprado);

      int nombreFichero=determinaNombreFichero(directorioComprados);


      peticionCompra.guardarEnArchivo(new
File(rutaDirectorioPeticionCompra+nombreFichero+".xml"));
      producto.guardarEnArchivo(new
File(rutaDirectorioProductoComprado+nombreFichero+".xml"));


      modeloComprados.insertaProducto(producto);
```

```
        System.out.println("COMPRA REALIZADA");
        JOptionPane.showMessageDialog(null,"ENHORABUENA, COMPRA
REALIZADA","Información de transacción realizada",
JOptionPane.INFORMATION_MESSAGE);

    }

//    Método que determina el nombre de un fichero XML con los datos del
//    producto en un determinado directorio. Todos los fichero llevan una
//    nomenclatura numérica, para determinar dicho número se lee el nombre de
//    todos los ficheros del directorio y se incrementa en 1 el mayor de ellos.
    int determinaNombreFichero(File directorio){

        Integer max=new Integer(0);
        Integer min;
        int comparacion;

        if(directorio.isDirectory()){

            File[] listaFicheros=directorio.listFiles();

            for(int i=0;i<listaFicheros.length;i++){

                String nombreFichero=listaFicheros[i].getName();
                String codigoFichero=nombreFichero.substring(0,nombreFichero.length()-4);
                min=new Integer(codigoFichero);
                comparacion=min.compareTo(max);
                if(comparacion>0){
                    max=min;
                }
            }
        }
        else {
            System.out.println("La ruta especificada no es un directorio");
        }

        return max.intValue()+1;
    }
}
```

1.1.4. Clase Vender

```
//
//    Fichero: Vender.java
//    Autor: Trinitario Meseguer Calero
//    Fecha: 08/03/05
//    Descripción: Esta clase interactúa con la clase ServidorServiciosJXTA,
//        ya que al recibir esta última un mensaje de petición de compra
//        se instancia un objeto de la clase Vender.Se encargará de enviar
//        la respuesta de perfil y la respuesta de compra del protocolo cXML
```

```
//


package PFC.codigo.acciones;

import PFC.codigo.jxta.*;
import PFC.codigo.tablas.*;
import PFC.codigo.utiles.*;
import PFC.codigo.xml.*;

import java.io.*;
import net.jxta.util.*;
import org.jdom.*;


public class Vender {
   JxtaBiDiPipe bipipe;

   String rutaDirectorioProductosVendidos="./Vendidos/";
   String rutaDirectorioProductosAnunciados="./Anuncios/";

   ModeloTablaVendidos modeloVendidos;
   ProductoXML producto;

   /** Creates a new instance of Vender */
   public Vender(JxtaBiDiPipe bipipe, ProductoXML producto, ModeloTablaVendidos
modeloVendidos) {
      this.bipipe=bipipe;
      this.modeloVendidos=modeloVendidos;
      this.producto=producto;
   }

//    Método que se encarga de rellenar y enviar la respuesta de perfil del
//    protocolo cXML
     public void enviarRespuestaPerfil(){

        RespuestaCXML respuestaPerfil=new RespuestaCXML("200","OK","");

        Element profileResponse=new Element("ProfileResponse");
        profileResponse.setAttribute("effectiveDate",FechaISO8601.fechaISO());

        Element transaction=new Element("Transaction");
        transaction.setAttribute("requestName","OrderRequest");

        profileResponse.addContent(transaction);

        respuestaPerfil.setTipoRespuesta(profileResponse);

        respuestaPerfil.enviar(bipipe, "PROFILE_RESPONSE");
```

```
    }

//   Método que se encarga de rellenar y enviar la respuesta de compra del
//   protocolo cXML
    public void enviarRespuestaCompra(){

       RespuestaCXML respuestaCompra=new RespuestaCXML("200","OK","");
       respuestaCompra.enviar(bipipe,"ORDER_RESPONSE");

    }

//   Método responsable de traspasar el fichero de un producto anunciado al
//   directorio de productos vendidos cuando la venta se ha realizado, así como
//   de insertar dicho producto en la tabla del panel de productos vendidos.
    public void ventaRealizada(String nombreFichero){


       File ficheroAnunciado=new
File(rutaDirectorioProductosAnunciados+nombreFichero);
       File ficheroVendido=new
File(rutaDirectorioProductosVendidos+nombreFichero);
       ficheroAnunciado.renameTo(ficheroVendido);
       modeloVendidos.insertaProducto(producto);
       System.out.println("VENTA REALIZADA");

    }
}
```

1.2. PAQUETE PFC.CODIGO.XML

1.2.1. Clase DatosDelComprador

```
//
//   Fichero: DatosComprador.java
//   Autor: Trinitario Meseguer Calero
//   Fecha: 15/01/05
//   Descripción: Esta clase se instancia en el momento de pulsar el
//      botón “Comprar” que aparece en el panel que muestra los
//      datos completos del producto encontrado que seleccionamos. Esta
//      clase hereda de la clase JFrame del paquete Swing. Es la
//      responsable del panel que recoge todos lo datos de compra.
//



package PFC.codigo.xml;

import PFC.codigo.acciones.*;
import PFC.codigo.jxta.*;
```

```
import PFC.codigo.tablas.*;
import PFC.codigo.utiles.*;

import java.io.*;
import net.jxta.util.*;
import java.awt.event.*;

public class DatosComprador extends javax.swing.JFrame {

    ProductoXML producto;
     ModeloTablaComprados modeloComprados;
     TuberiaCliente tuberia;


    /** Creates new form DatosComprador */
    public DatosComprador(ProductoXML producto, TuberiaCliente tuberia,
ModeloTablaComprados modeloComprados) {
        this.producto=producto;
        this.modeloComprados=modeloComprados;
        initComponents();
        this.tuberia=tuberia;

    }

//   Método que inicializa los componentes gráficos de esta clase, que se
//   corresponden con el panel donde el comprador debe introducir sus datos
    private void initComponents() {
        panelDatosCompra = new javax.swing.JPanel();
        panelFormaPago = new javax.swing.JPanel();
        opcFormaPago = new javax.swing.JComboBox();
        etNumTarjeta = new javax.swing.JLabel();
        campoNumTarjeta = new javax.swing.JTextField();
        etCaducidad = new javax.swing.JLabel();
        campoCaducidad = new javax.swing.JTextField();
        panelFacturacion = new javax.swing.JPanel();
        etNombre = new javax.swing.JLabel();
        etNIF = new javax.swing.JLabel();
        etDireccion = new javax.swing.JLabel();
        etLocalidad = new javax.swing.JLabel();
        etCP = new javax.swing.JLabel();
        etProvincia = new javax.swing.JLabel();
        etPais = new javax.swing.JLabel();
        campoNombre = new javax.swing.JTextField();
        campoDireccion = new javax.swing.JTextField();
        campoCP = new javax.swing.JTextField();
        campoLocalidad = new javax.swing.JTextField();
        campoProvincia = new javax.swing.JTextField();
        campoPais = new javax.swing.JTextField();
        campoNIF = new javax.swing.JTextField();
        PanelEnvio = new javax.swing.JPanel();
```

```
        etAtencion = new javax.swing.JLabel();
        etDireccionEnv = new javax.swing.JLabel();
        etLocalidadEnv = new javax.swing.JLabel();
        etProvinciaEnv = new javax.swing.JLabel();
        etCPEnv = new javax.swing.JLabel();
        etPaisEnv = new javax.swing.JLabel();
        etTelefonoEnv = new javax.swing.JLabel();
        campoAtencion = new javax.swing.JTextField();
        campoDireccionEnv = new javax.swing.JTextField();
        campoTelefonoEnv = new javax.swing.JTextField();
        campoLocalidaEnv = new javax.swing.JTextField();
        campoProvinciaEnv = new javax.swing.JTextField();
        campoPaisEnv = new javax.swing.JTextField();
        campoCPEnv = new javax.swing.JTextField();
        EtDatosCompra = new javax.swing.JLabel();
        BotonComprar = new javax.swing.JButton();
        panelDatosCXML = new javax.swing.JPanel();
        etNombreDominio = new javax.swing.JLabel();
        etIdentidadDominio = new javax.swing.JLabel();
        etClave = new javax.swing.JLabel();
        campoNombreDominio = new javax.swing.JTextField();
        campoIdentidadDominio = new javax.swing.JTextField();
        campoClave = new javax.swing.JTextField();

        setTitle("Datos de Compra");
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                exitForm(evt);
            }
        });

        panelDatosCompra.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        panelDatosCompra.setFont(new java.awt.Font("MS Sans Serif", 0, 10));
        panelFormaPago.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        panelFormaPago.setBorder(new javax.swing.border.TitledBorder("Forma de
Pago:"));
        panelFormaPago.setFont(new java.awt.Font("Century Gothic", 1, 12));
        opcFormaPago.setModel(new javax.swing.DefaultComboBoxModel(new String[]
{ "PCard" }));
        panelFormaPago.add(opcFormaPago, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 30, -1, -1));

        etNumTarjeta.setText("N\u00fam. Tarjeta");
        panelFormaPago.add(etNumTarjeta, new
org.netbeans.lib.awtextra.AbsoluteConstraints(90, 30, -1, -1));

        panelFormaPago.add(campoNumTarjeta, new
org.netbeans.lib.awtextra.AbsoluteConstraints(180, 30, 140, 20));
```

```
        etCaducidad.setText("Caducidad");
        panelFormaPago.add(etCaducidad, new
org.netbeans.lib.awtextra.AbsoluteConstraints(350, 30, -1, -1));

        panelFormaPago.add(campoCaducidad, new
org.netbeans.lib.awtextra.AbsoluteConstraints(420, 30, 90, -1));

        panelDatosCompra.add(panelFormaPago, new
org.netbeans.lib.awtextra.AbsoluteConstraints(30, 320, 530, 80));

        panelFacturacion.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        panelFacturacion.setBorder(new javax.swing.border.TitledBorder("Facturar a:"));
        etNombre.setText("Nombre/Raz\u00f3n Social");
        panelFacturacion.add(etNombre, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 20, -1, -1));

        etNIF.setText("NIF/CIF");
        panelFacturacion.add(etNIF, new
org.netbeans.lib.awtextra.AbsoluteConstraints(340, 20, -1, -1));

        etDireccion.setText("Direcci\u00f3n");
        panelFacturacion.add(etDireccion, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 50, -1, -1));

        etLocalidad.setText("Localidad");
        panelFacturacion.add(etLocalidad, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 80, -1, -1));

        etCP.setText("C.P.");
        panelFacturacion.add(etCP, new
org.netbeans.lib.awtextra.AbsoluteConstraints(340, 50, -1, -1));

        etProvincia.setText("Provincia:");
        panelFacturacion.add(etProvincia, new
org.netbeans.lib.awtextra.AbsoluteConstraints(180, 80, -1, -1));

        etPais.setText("Pais:");
        panelFacturacion.add(etPais, new
org.netbeans.lib.awtextra.AbsoluteConstraints(340, 80, -1, -1));

        panelFacturacion.add(campoNombre, new
org.netbeans.lib.awtextra.AbsoluteConstraints(160, 21, 160, 20));

        panelFacturacion.add(campoDireccion, new
org.netbeans.lib.awtextra.AbsoluteConstraints(80, 50, 220, 20));

        panelFacturacion.add(campoCP, new
org.netbeans.lib.awtextra.AbsoluteConstraints(370, 50, 90, 20));
```

```
        panelFacturacion.add(campoLocalidad, new
org.netbeans.lib.awtextra.AbsoluteConstraints(80, 80, 90, 20));

        panelFacturacion.add(campoProvincia, new
org.netbeans.lib.awtextra.AbsoluteConstraints(240, 80, 90, 20));

        panelFacturacion.add(campoPais, new
org.netbeans.lib.awtextra.AbsoluteConstraints(370, 80, 90, 20));

        panelFacturacion.add(campoNIF, new
org.netbeans.lib.awtextra.AbsoluteConstraints(380, 20, 90, 20));

        panelDatosCompra.add(panelFacturacion, new
org.netbeans.lib.awtextra.AbsoluteConstraints(30, 40, 530, 120));

        PanelEnvio.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        PanelEnvio.setBorder(new javax.swing.border.TitledBorder("Enviar a:"));
        PanelEnvio.setFont(new java.awt.Font("Century Gothic", 1, 12));
        etAtencion.setText("A la atenci\u00f3n de:");
        PanelEnvio.add(etAtencion, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 20, -1, -1));

        etDireccionEnv.setText("Direcci\u00f3n");
        PanelEnvio.add(etDireccionEnv, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 50, -1, -1));

        etLocalidadEnv.setText("Localidad");
        PanelEnvio.add(etLocalidadEnv, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 80, -1, -1));

        etProvinciaEnv.setText("Provincia:");
        PanelEnvio.add(etProvinciaEnv, new
org.netbeans.lib.awtextra.AbsoluteConstraints(180, 80, -1, -1));

        etCPEnv.setText("C.P.");
        PanelEnvio.add(etCPEnv, new org.netbeans.lib.awtextra.AbsoluteConstraints(340,
50, -1, -1));

        etPaisEnv.setText("Pais:");
        PanelEnvio.add(etPaisEnv, new
org.netbeans.lib.awtextra.AbsoluteConstraints(340, 80, -1, -1));

        etTelefonoEnv.setText("Telefono");
        PanelEnvio.add(etTelefonoEnv, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 110, -1, -1));

        PanelEnvio.add(campoAtencion, new
org.netbeans.lib.awtextra.AbsoluteConstraints(120, 20, 240, 20));
```

```
        PanelEnvio.add(campoDireccionEnv, new
org.netbeans.lib.awtextra.AbsoluteConstraints(80, 50, 180, 20));

        PanelEnvio.add(campoTelefonoEnv, new
org.netbeans.lib.awtextra.AbsoluteConstraints(80, 110, 90, 20));

        PanelEnvio.add(campoLocalidaEnv, new
org.netbeans.lib.awtextra.AbsoluteConstraints(80, 80, 90, 20));

        PanelEnvio.add(campoProvinciaEnv, new
org.netbeans.lib.awtextra.AbsoluteConstraints(240, 80, 90, 20));

        PanelEnvio.add(campoPaisEnv, new
org.netbeans.lib.awtextra.AbsoluteConstraints(370, 80, 90, 20));

        PanelEnvio.add(campoCPEnv, new
org.netbeans.lib.awtextra.AbsoluteConstraints(370, 50, 90, 20));

        panelDatosCompra.add(PanelEnvio, new
org.netbeans.lib.awtextra.AbsoluteConstraints(30, 170, 530, 140));

        EtDatosCompra.setFont(new java.awt.Font("Century Gothic", 1, 14));
        EtDatosCompra.setText("DATOS DE COMPRA");
        panelDatosCompra.add(EtDatosCompra, new
org.netbeans.lib.awtextra.AbsoluteConstraints(230, 10, -1, -1));

        BotonComprar.setText("COMPRAR");
        BotonComprar.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                BotonCompararActionPerformed(evt);
            }
        });

        panelDatosCompra.add(BotonComprar, new
org.netbeans.lib.awtextra.AbsoluteConstraints(230, 500, -1, -1));

        panelDatosCXML.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        panelDatosCXML.setBorder(new javax.swing.border.TitledBorder("Informacion
de la Red \"Comercio Electronico 1.0\""));
        etNombreDominio.setText("Nombre de Dominio:");
        panelDatosCXML.add(etNombreDominio, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 30, -1, -1));

        etIdentidadDominio.setText("Identidad de Dominio:");
        panelDatosCXML.add(etIdentidadDominio, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 60, 140, -1));

        etClave.setText("Clave:");
```

```
        panelDatosCXML.add(etClave, new
org.netbeans.lib.awtextra.AbsoluteConstraints(320, 60, -1, -1));

        panelDatosCXML.add(campoNombreDominio, new
org.netbeans.lib.awtextra.AbsoluteConstraints(180, 30, 110, -1));

        panelDatosCXML.add(campoIdentidadDominio, new
org.netbeans.lib.awtextra.AbsoluteConstraints(180, 60, 110, -1));

        panelDatosCXML.add(campoClave, new
org.netbeans.lib.awtextra.AbsoluteConstraints(380, 60, 100, -1));

        panelDatosCompra.add(panelDatosCXML, new
org.netbeans.lib.awtextra.AbsoluteConstraints(30, 400, 530, 90));

        getContentPane().add(panelDatosCompra, java.awt.BorderLayout.CENTER);

        pack();
    }


//    Método llamado al pulsar el botón "Comprar" del panel con los datos del
//    comprador y que recoge todos los datos introducidos para rellenar un
//    documento de petición de compra
    private void BotonComprarActionPerformed(java.awt.event.ActionEvent evt) {

        PeticionDeCompraCXML peticionDeCompra=new PeticionDeCompraCXML();

        peticionDeCompra.setClaveSender(campoClave.getText());

peticionDeCompra.setDominioCredencialFrom(campoNombreDominio.getText());

peticionDeCompra.setDominioCredencialSender(campoNombreDominio.getText());
        peticionDeCompra.setIdentidadFrom(campoIdentidadDominio.getText());
        peticionDeCompra.setIdentidadSender(campoIdentidadDominio.getText());

        peticionDeCompra.setNumeroTarjeta(campoNumTarjeta.getText());
        peticionDeCompra.setFechaCaducidad(campoCaducidad.getText());

        peticionDeCompra.setCPFacturacion(campoCP.getText());
        peticionDeCompra.setDireccionFacturacion(campoDireccion.getText());
        peticionDeCompra.setNombreFacturacion(campoNombre.getText());
        peticionDeCompra.setLocalidadFacturacion(campoLocalidad.getText());
        peticionDeCompra.setProvinciaFacturacion(campoProvincia.getText());
        peticionDeCompra.setPaisFacturacion(campoPais.getText());

        peticionDeCompra.setCPEnvio(campoCPEnv.getText());
        peticionDeCompra.setDireccionEnvio(campoDireccionEnv.getText());
        peticionDeCompra.setNombreEnvio(campoAtencion.getText());
        peticionDeCompra.setLocalidadEnvio(campoLocalidaEnv.getText());
```

```
        peticionDeCompra.setProvinciaEnvio(campoProvinciaEnv.getText());
        peticionDeCompra.setPaisEnvio(campoPaisEnv.getText());

        peticionDeCompra.setDatosProducto(producto);

        Comprar compra=new
Comprar(peticionDeCompra,producto,tuberia,modeloComprados);

        compra.enviarPeticionPerfil();

         WindowEvent eventoCierreVentana=new WindowEvent(this,
WindowEvent.WINDOW_CLOSING);
        this.dispatchEvent(eventoCierreVentana);


    }

//    Devuelve el control del programa al objeto que lo llamó
    private void exitForm(java.awt.event.WindowEvent evt) {
        return;
    }



    // Variables declaration - do not modify
    private javax.swing.JButton BotonComprar;
    private javax.swing.JLabel EtDatosCompra;
    private javax.swing.JPanel PanelEnvio;
    private javax.swing.JTextField campoAtencion;
    private javax.swing.JTextField campoCP;
    private javax.swing.JTextField campoCPEnv;
    private javax.swing.JTextField campoCaducidad;
    private javax.swing.JTextField campoClave;
    private javax.swing.JTextField campoDireccion;
    private javax.swing.JTextField campoDireccionEnv;
    private javax.swing.JTextField campoIdentidadDominio;
    private javax.swing.JTextField campoLocalidaEnv;
    private javax.swing.JTextField campoLocalidad;
    private javax.swing.JTextField campoNIF;
    private javax.swing.JTextField campoNombre;
    private javax.swing.JTextField campoNombreDominio;
    private javax.swing.JTextField campoNumTarjeta;
    private javax.swing.JTextField campoPais;
    private javax.swing.JTextField campoPaisEnv;
    private javax.swing.JTextField campoProvincia;
    private javax.swing.JTextField campoProvinciaEnv;
    private javax.swing.JTextField campoTelefonoEnv;
    private javax.swing.JLabel etAtencion;
    private javax.swing.JLabel etCP;
    private javax.swing.JLabel etCPEnv;
```

```
    private javax.swing.JLabel etCaducidad;
    private javax.swing.JLabel etClave;
    private javax.swing.JLabel etDireccion;
    private javax.swing.JLabel etDireccionEnv;
    private javax.swing.JLabel etIdentidadDominio;
    private javax.swing.JLabel etLocalidad;
    private javax.swing.JLabel etLocalidadEnv;
    private javax.swing.JLabel etNIF;
    private javax.swing.JLabel etNombre;
    private javax.swing.JLabel etNombreDominio;
    private javax.swing.JLabel etNumTarjeta;
    private javax.swing.JLabel etPais;
    private javax.swing.JLabel etPaisEnv;
    private javax.swing.JLabel etProvincia;
    private javax.swing.JLabel etProvinciaEnv;
    private javax.swing.JLabel etTelefonoEnv;
    private javax.swing.JComboBox opcFormaPago;
    private javax.swing.JPanel panelDatosCXML;
    private javax.swing.JPanel panelDatosCompra;
    private javax.swing.JPanel panelFacturacion;
    private javax.swing.JPanel panelFormaPago;
    // End of variables declaration

}
```

1.2.2. Clase Categorías

```
//
//   Fichero: Categorias.java
//   Autor: Trinitario Meseguer Calero
//   Fecha: 03/12/04
//   Descripción: La clase Categorías va a ser la responsable de leer
//      el archivo "categorias.xml" y extraer del mismo las categorías
//      y su correspondientes subcategorías.
//

package PFC.codigo.xml;

import PFC.codigo.acciones.*;
import PFC.codigo.jxta.*;
import PFC.codigo.tablas.*;
import PFC.codigo.utiles.*;

import org.jdom.input.SAXBuilder;
import org.jdom.*;
import java.util.*;

public class Categorias {

   SAXBuilder builder;
```

```
   org.jdom.Document docCat;
   Element raiz;
   public Element categorias[];

   /** Creates a new instance of Categorias */
   public Categorias() {

      try{
         builder = new SAXBuilder();
         docCat = builder.build("./codigo/categorias.xml");
         raiz=docCat.getRootElement();
         categorias=ExtraeSubelementos(raiz);
      }
      catch (Exception e) {

         System.out.println("Error interno " + e);

      }
   }

//   Método que extrae todos los elementos hijos de un elemento que se le pasa
//   como parámetro
   public static Element[] ExtraeSubelementos(Element elemento) throws
JDOMException {



         List listaSubelem=elemento.getChildren();
         int numSubelem=listaSubelem.size();
         Element subelementos[]=new Element[numSubelem];

         for(int i=0;i<numSubelem;i++){
            subelementos[i]=(Element)listaSubelem.get(i);

         }


      return subelementos;

   }

//   Método que extrae en forma de lista de String el contenido de una
//   lista de elementos
    public static String [] ExtraeContenidoElementos(Element listaelem[]){

        int numelem=listaelem.length;
        String listacad[]=new String[numelem];
        for(int i=0;i<numelem;i++){
           listacad[i]=listaelem[i].getTextNormalize();
```

```
        }
        return listacad;
    }

}
```

1.2.3. Clase DocumentoXML

```
//
//   Fichero: DocumentoXML.java
//   Autor: Trinitario Meseguer Calero
//   Fecha: 15/01/05
//   Descripción: Contiene la clase DocumentoXML, que representa a un documento
//          de contenido XML mediante una representación JDOM.
//

package PFC.codigo.xml;

import PFC.codigo.acciones.*;
import PFC.codigo.jxta.*;
import PFC.codigo.tablas.*;
import PFC.codigo.utiles.*;

import java.io.File;
import java.io.Serializable;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;

import java.util.List;

import org.jdom.*;
import org.jdom.input.SAXBuilder;
import org.jdom.output.XMLOutputter;
import org.jdom.xpath.XPath;

import net.jxta.util.*;
import net.jxta.endpoint.*;

/**
 * Esta clase representa a un objeto XML construido mediante JDOM.
 * Contiene una serie de métodos útiles para generar estructuras XML y
 * para poder acceder a los nodos JDOM.
 *
 *
 */
public class DocumentoXML extends    Object
                    implements Serializable,
```

```
                        Cloneable {

private final static String SenderMessage = "ComercioP2P";
/**
 * Almacena un documento XML de JDOM
 */
 public Document documento;

/**
 * Constructor básico de la clase. Crea una nueva instancia de
 * <code>DocumentoXML</code> vacía.
 */
public DocumentoXML() {
   documento = new Document();
}

/**
 * Crea una nueva instancia de <code>DocumentoXML</code> a partir de un
 * documento JDOM.
 *
 */
public DocumentoXML(Document documento) {
   this.documento = documento;
}

/**
 * Crea una nueva instancia de <code>DocumentoXML</code> a partir de un
 * archivo que debe contener un texto XML válido cualesquiera.
 *
 */
public DocumentoXML(File archivo) throws IOException {
   //Se crea un constructor de árboles JDOM basado en SAX.
   SAXBuilder constructorSAX = new SAXBuilder();
   /*
    * No se valida puesto que si la DTD no está disponible, se
    * produciría un error.
    */
   constructorSAX.setValidation(false);

   try {
      //Se crea un nuevo objeto Document de JDOM a partir del archivo.
      documento = constructorSAX.build(archivo);
   } catch (JDOMException je) {
      documento = null; //Se capturan los errores de parseo
   } catch (IOException ioe) {
      throw ioe;       //pero no los relacionados con I/O del fichero
   }
}

/**
```

```
 * Crea una nueva instancia de <code>DocumentoXML</code> e inicializa
 * la declaración de tipo de documento XML con los valores
 * especificados en <code>tipo</code> y <code>dtd</code>.<br>
 *
 */
public DocumentoXML(String tipo, String dtd) {
   this();

   //Se crea el elemento de tipo de documento XML
   DocType doctype = new DocType(tipo);
   doctype.setSystemID(dtd);

   documento.setDocType(doctype); //se asigna al documento JDOM
}

/**
 * Obtiene una representación de este objeto <code>DocumentoXML</code>
 * en forma de cadena <code>String</code> en formato XML.
 *
 */
public String toString() {
   //Se crea un objeto que transforma árboles JDOM en flujos XML.
   XMLOutputter out = new XMLOutputter();
   out.setNewlines(true); //para separar las nuevas líneas
      out.setIndent("   ");  //para que haya indentación
   //El JDOM se convierte en flujo XML en forma de String
   return out.outputString(documento);
}

/**
 * Crea una nueva instancia de <code>DocumentoXML</code> idéntica
 * a este objeto.
 *
 */
public Object clone() {
   return (Object) new DocumentoXML(documento);
}

/**
 * Determina si el objeto especificado es igual a este.
 *
 */
public boolean equals(Object objeto) {
   return documento.equals(objeto);
}

/**
 * Almacena el documento XML representado por este objeto en el archivo
 * especificado. El texto XML se guardará como texto formateado e
 * indentado.
```

```
 *
 */
public boolean guardarEnArchivo(File archivo) {
   boolean resultado;

   try {
      //Se crea un objeto que transforma árboles JDOM en flujos

      XMLOutputter out = new XMLOutputter();
      //Se crea un flujo de escritura hacia el archivo
      OutputStream flujo = new FileOutputStream(archivo);

      out.setNewlines(true); //para separar las nuevas líneas
      out.setIndent("   ");  //para que haya indentación
      //Se escribe en el archivo el flujo XML
      out.output(documento, flujo);
      flujo.close();

      resultado = true;
   } catch(Exception ex) {
      //No se procesa el error, pero se devuelve false
      resultado = false;

   }

   return resultado;
}

/**
 * Devuelve un documento <code>Document</code> con la representación
 * en JDOM del XML de este objeto.
 *
 */
public Document getDocumento() {
   return documento;
}

/**
 * Establece el contenido XML del documento <code>Document</code> JDOM
 * especificado en este objeto.
 *
 */
public void setDocumento(Document documento) {
   this.documento = documento;
}

/**
 * Devuelve el elemento raiz en forma de elemento JDOM de este
 * documento XML.
 *
```

```
 */
public Element getElementoRaiz() {
   return documento.getRootElement();
}

/**
 * Establece el elemento raiz de este documento XML a partir de
 * un elemento JDOM
 *
 */
public void setElementoRaiz(Element raiz) {
   documento.setRootElement(raiz);
}

/**
 * Obtiene la declaración de tipo de documento XML de este documento.
 *
 */
public DocType getTipoDocumento() {
   return documento.getDocType();
}

/**
 * Obtiene un sólo objeto JDOM a partir de una ruta válida XPATH.
 * Este método es una implementación básica de XPATH proporcionado
 * por JDOM.
 *
 */
public Object getObjetoPorXPath(String ruta) {
   Object resultado = null;
   try {
      XPath xpath = XPath.newInstance(ruta);
      /*
       * Se selecciona un solo elemento XML, que depende de la
       * expresión XPATH usada.
       */
      resultado = xpath.selectSingleNode(documento);
   } catch (JDOMException jde) {
      resultado = null;
   }
   return resultado;
}

/**
 * Obtiene una lista de objetos JDOM a partir de una ruta válida XPATH.
 * Este método es una implementación básica de XPATH proporcionado
 * por JDOM.
 *
 */
public List getObjetosPorXPath(String ruta) {
```

```
    List resultado = null;
    try {
        XPath xpath = XPath.newInstance(ruta);
        /*
         * Se selecciona varios elementos XML, dependiendo de la
         * expresión XPATH usada.
         */
        resultado = xpath.selectNodes(documento);
    } catch (JDOMException jde) {
        resultado = null;
    }
    return resultado;
}

/**
 * Obtiene un elemento <code>Element</code> JDOM a partir de la ruta
 * XPATH especificada.
 *
 */
public Element getElemento(String ruta) {
    return (Element) getObjetoPorXPath(ruta);
}

/**
 * Obtiene una cadena con el valor de un atributo de un elemento XML,
 * dado una ruta XPATH válida hacia el elemento y el nombre del atributo
 * cuyo valor se quiere obtener.
 *
 */
public String getValorAtributo(String ruta, String atributo) {
    Attribute at = (Attribute) getObjetoPorXPath(ruta + "/@" + atributo);
    return at.getValue();
}

/**
 * Devuelve si el documento XML representado por este objeto es
 * bien formado respecto a la sintaxis XML.
 *
 */
public boolean esBienFormado() {
    boolean resultado = false;
    InputStream is;

    if( documento != null) {
        is = convertirEnInputStream();
        resultado = AnalizadorXML.esBienFormado(is);
    }

    return resultado;
}
```

```
/**
 * Devuelve si el documento XML representado por este objeto es
 * válido respecto al DTD referenciado en la declaración de tipo
 * de documento XML incluida en el documenti XML.
 *
 */
public boolean esValido() {
   boolean resultado = false;
   InputStream is;

   if( documento != null) {
      is = convertirEnInputStream();
      resultado = AnalizadorXML.esValido(is);
   }

   return resultado;
}

/**
 * Convierte el documento XML representado por este objeto en un flujo
 * de entrada <code>InputStream</code>.
 *
 */
private InputStream convertirEnInputStream() {
   InputStream resultado;
   //Se crea un objeto que transforma árboles JDOM en flujos de bytes
   XMLOutputter out = new XMLOutputter();

   if( documento != null) {
      try {
         //Se crea el flujo de salida
         ByteArrayOutputStream os = new ByteArrayOutputStream();
         //Se escribe del XML al flujo de salida
         out.output(documento, os);

         resultado = new ByteArrayInputStream(os.toByteArray());
      } catch (IOException ioe) {
         resultado = null;
      }

   } else {
      resultado = null;
   }
   return resultado;
}

//   Método que envía el DocumentoXML a través de un pipe bidireccional
//   encapsulado en un mensaje JXTA cuyo tipo se le pasa como parámetro
   public void enviar(JxtaBiDiPipe bipipe, String tipoMensaje){
```

```
      try {

         Message msg = new Message();

         msg.addMessageElement("tipo",
                        new StringMessageElement("tipo",
                                                 tipoMensaje,
                                                 null));
         msg.addMessageElement(SenderMessage,
                         new StringMessageElement(SenderMessage,
                                                 this.toString(),
                                                 null));

         bipipe.sendMessage(msg);
      }
       catch (Exception ie) {
         ie.printStackTrace();
      }

   }
}
```

1.2.4. Clase ProductoXML

```
//
//   Fichero: ProductoXML.java
//   Autor: Trinitario Meseguer Calero
//   Fecha: 20/01/05
//   Descripción: Se trata de una subclase de la clase DocumentoXML,
//      por lo tanto posee todos sus métodos. Se trata de una clase
//      fundamental en nuestra aplicación pues todo el manejo de productos
//      se realiza mediante objetos de esta clase. Al instanciar
//      esta clase se genera un objeto JDOM en el que se almacena toda
//      la información relativa de un producto, es decir, datos del vendedor,
//      datos de la red “Comercio Electrónico 1.0” y datos del producto.
//



package PFC.codigo.xml;

import PFC.codigo.acciones.*;
import PFC.codigo.jxta.*;
import PFC.codigo.tablas.*;
import PFC.codigo.utiles.*;

import org.jdom.*;
import org.jdom.input.*;
import org.jdom.output.XMLOutputter;
```

```
import java.io.*;


public class ProductoXML extends DocumentoXML {

    public Element root;
    public Element datosProveedor;
    public Element datosProducto;
    public Element nombre;
    public Element apellidos;
    public Element direccion;
    public Element localidad;
    public Element provincia;
    public Element cp;
    public Element telefono;
    public Element dni;
    public Element cif;
    public Element empresa;
    public Element codigoProd;
    public Element categoria;
    public Element subcategoria;
    public Element nombreProducto;
    public Element descripcion;
    public Element precio;
    public Element fechaAnuncio;


    public Element datosRed;
    public Element identidad;
    public Element dominio;


//    Constructor de la clase ProductoXML y que establece la estructura
//    de dicho objeto
    public ProductoXML() {

        super();

        root = new Element("Producto");
        datosProveedor=new Element("DatosProveedor");
        datosProducto=new Element("DatosProducto");
        datosRed=new Element("DatosRed");

        root.addContent(datosProveedor);
        root.addContent(datosProducto);
        root.addContent(datosRed);


        nombre=new Element("Nombre");
        apellidos=new Element("Apellidos");
```

```
        direccion=new Element("Direccion");
        localidad=new Element("Localidad");
        provincia=new Element("Provincia");
        cp=new Element("CP");
        telefono=new Element("Telefono");
        dni=new Element("DNI");
        cif=new Element("CIF");
        empresa=new Element("Empresa");

        datosProveedor.addContent(nombre);
        datosProveedor.addContent(apellidos);
        datosProveedor.addContent(dni);
        datosProveedor.addContent(telefono);
        datosProveedor.addContent(empresa);
        datosProveedor.addContent(cif);
        datosProveedor.addContent(direccion);
        datosProveedor.addContent(localidad);
        datosProveedor.addContent(provincia);
        datosProveedor.addContent(cp);

        codigoProd=new Element("CodigoProducto");
        categoria=new Element("Categoria");
        subcategoria=new Element("Subcategoria");
        nombreProducto=new Element("NombreProducto");
        descripcion=new Element("Descripcion");
        precio=new Element("Precio");
        fechaAnuncio=new Element("FechaAnuncio");

        datosProducto.addContent(codigoProd);
        datosProducto.addContent(categoria);
        datosProducto.addContent(subcategoria);
        datosProducto.addContent(nombreProducto);
        datosProducto.addContent(descripcion);
        datosProducto.addContent(precio);
        datosProducto.addContent(fechaAnuncio);

        identidad=new Element("Identity");
        dominio=new Element("Domain");

        datosRed.addContent(identidad);
        datosRed.addContent(dominio);

       this.setElementoRaiz(root);

    }

//    Constructor de la clase ProductoXML que iniciliza un objeto de esta clase
//    a partir de un fichero que se le pasa como argumento
    public ProductoXML(File fichero) throws IOException {
```

```
      super(fichero);
       try {


          root=documento.getRootElement();
          datosProveedor=root.getChild("DatosProveedor");
          datosProducto=root.getChild("DatosProducto");
          datosRed=root.getChild("DatosRed");

          nombre=datosProveedor.getChild("Nombre");
          apellidos=datosProveedor.getChild("Apellidos");
          direccion=datosProveedor.getChild("Direccion");
          localidad=datosProveedor.getChild("Localidad");
          provincia=datosProveedor.getChild("Provincia");
          cp=datosProveedor.getChild("CP");
          telefono=datosProveedor.getChild("Telefono");
          dni=datosProveedor.getChild("DNI");
          cif=datosProveedor.getChild("CIF");
          empresa=datosProveedor.getChild("Empresa");

          codigoProd=datosProducto.getChild("CodigoProducto");
          categoria=datosProducto.getChild("Categoria");
          subcategoria=datosProducto.getChild("Subcategoria");
          nombreProducto=datosProducto.getChild("NombreProducto");
          descripcion=datosProducto.getChild("Descripcion");
          precio=datosProducto.getChild("Precio");
          fechaAnuncio=datosProducto.getChild("FechaAnuncio");


          dominio=datosRed.getChild("Domain");
          identidad=datosRed.getChild("Identity");

       }
      catch (Exception e) {
        System.err.println(e);
      }

   }

//    Método que inicializa los datos relativos al proveedor de un objeto de
//    esta clase
    public void RellenarDatosProveedor (String nombre,String apellidos, String
direccion, String localidad, String provincia, String cp, String telefono, String dni,
String cif, String empresa){



      this.nombre.addContent(nombre);
      this.apellidos.addContent(apellidos);
      this.direccion.addContent(direccion);
```

```
        this.localidad.addContent(localidad);
        this.provincia.addContent(provincia);
        this.cp.addContent(cp);
        this.telefono.addContent(telefono);
        this.dni.addContent(dni);
        this.cif.addContent(cif);
        this.empresa.addContent(empresa);

    }

//    Método que inicializa los datos relativos al producto de un objeto de
//    esta clase
    public void RellenarDatosProducto(String codigoProd,String categoria, String
subcategoria, String nombreProducto, String descripcion, String precio, String
fechaAnuncio){

        this.codigoProd.addContent(codigoProd);
        this.categoria.addContent(categoria);
        this.subcategoria.addContent(subcategoria);
        this.nombreProducto.addContent(nombreProducto);
        this.descripcion.addContent(descripcion);
        this.precio.addContent(precio);
        this.fechaAnuncio.addContent(fechaAnuncio);

    }

//    Método que inicializa los datos relativos a la red "Comercio Electrónico"
//    de un objeto de esta clase
    public void RellenarDatosRed(String identidad, String dominio){
        this.identidad.addContent(identidad);
        this.dominio.addContent(dominio);

    }

//    Método que devuelve el contenido del objeto ProductoXML en forma de
//    String
    public String toString() {

        //Se crea un objeto que transforma árboles JDOM en flujos XML.
        XMLOutputter out = new XMLOutputter();
        out.setNewlines(true); //para separar las nuevas líneas
        //El JDOM se convierte en flujo XML en forma de String
        return out.outputString(documento);
    }
}
```

1.2.5. Clase DocumentoCXML

```
//
//    Fichero: DocumentoCXML.java
//    Autor: Trinitario Meseguer Calero
```

```
//    Fecha: 15/01/05
//    Descripción: Contiene la clase DocumentoCXML, que representa un documento
//                 de contenido cXML de propósito general.
//

package PFC.codigo.xml;

import PFC.codigo.acciones.*;
import PFC.codigo.jxta.*;
import PFC.codigo.tablas.*;
import PFC.codigo.utiles.*;

import java.io.File;
import java.io.IOException;

import org.jdom.Element;



/**
 * Representa a un documento XML con contenido cXML (commerce XML).
 * El tipo de documentos cXML representados pueden ser: <br>
 * 1.- cXML     : peticiones y respuestas de órdenes de compra, transacciones
 *                punchout, facturas, etc. <br>
 * 2.- Supplier : para almacenar información acerca de un proveedor. <br>
 * 3.- Index    : para almacenar información acerca de un catálogo. <br>
 * Todos los documentos cXML deben ser validados respecto del DTD
 * proporcionado por <a href="www.cxml.org">cXML.org</a>. <br><br>
 * <b>Nota:</b> <br>
 * <i>Esta implementación usa un DTD local a la máquina, de forma que la
 * validación del documento sea más rápida. Si se usara el DTD externo, el
 * parser XML debe acceder a ese recurso con el incremento de tiempo de
 * proceso que ello requiere, y con el peligro de no estar disponible en
 * ese momento el recurso, bien sea por falta de conexión, o por
 * imposibilidad de encontrar el recurso. Se debe asegurar que el DTD usado
 * por cXML, "cXML.dtd", se encuentre disponible en los directorios
 * de trabajo. <i>
 *
 *
 */
public class DocumentoCXML extends DocumentoXML {



   /**
    * Constante que define la localización relativa en la propia máquina
    * del DTD usado para validar los documentos CXML.
    */
   protected final static String CXML_DTD = "cXML.dtd";
```

```
/**
 * Constante que define la localización URL del DTD usado para validar
 * los documentos CXML.
 */
protected final static String CXML_DTD_URL =
                "http://xml.cXML.org/schemas/cXML/1.2.009/cXML.dtd";

/**
 * Constante que define el tipo "cXML" para la declaración de tipo de
 * documento XML, usado para la mayoría de documentos CXML que deben
 * cumplir con el DTD de cXML.
 */
public final static String TIPO_CXML = "cXML";

/**
 * Constante que define el tipo "Index" para la declaración de tipo de
 * documento XML, usado para los documentos de catálogo cXML.
 */
public final static String TIPO_CATALOGO = "Index";

/**
 * Constante que define el tipo "Supplier" para la declaración de tipo de
 * documento XML, usado para los documentos de datos de proveedores cXML.
 */
public final static String TIPO_DATOS = "Supplier";

/**
 * Constructor básico de la clase. Crea un nuevo objeto
 * <code>DocumentoCXML</code> vacío.
 */
public DocumentoCXML() {
   super();
}

/**
 * Crea una nueva instancia de esta clase a partir del XML contenido
 * en el archivo especificado. Este constructor debe ser usado con
 * precaución, puesto que no comprueba que el documento XML sea
 * realmente de contenido cXML, por lo que se debe asegurar que el
 * XML sea de tipo cXML. El documento cXML será validado frente al
 * DTD que tenga en la declaración de tipo de documento XML.
 *
 */
public DocumentoCXML(File archivo) throws IOException {
   super(archivo);
}

/**
 * Crea una nueva instancia de esta clase, creando una nueva estructura
 * XML mediante la el establecimiento de la declaración de tipo de
```

```
     * documento XML y del elemento raíz.
     *
     */
    public DocumentoCXML(String tipo) {
        super(tipo, CXML_DTD);
        Element raiz = new Element(tipo);
        setElementoRaiz(raiz);
    }

    /**
     * Crea un objeto de esta clase idéntico a este objeto.
     *
     */
    public Object clone() {
        DocumentoCXML doc = new DocumentoCXML();
        doc.setDocumento(documento);
        return doc;
    }

}
```

1.2.6. Clase PeticionCXML

```
//
//    Fichero: PeticionCXML.java
//    Autor: Trinitario Meseguer Calero
//    Fecha: 15/01/05
//    Descripción: Contiene la clase PeticionCXML, que representa a un documento
//              de contenido cXML de petición.
//


package PFC.codigo.xml;

import PFC.codigo.acciones.*;
import PFC.codigo.jxta.*;
import PFC.codigo.tablas.*;
import PFC.codigo.utiles.*;

import java.util.List;
import org.jdom.*;


/**
 * Clase que representa un documento de petición basado en cXML. <br>
 * En el modelo de intercambio de mensajes de comercio eléctronico
 * propuesto por cXML, los clientes envían peticiones a los concentradores
 * de comercio, como por ejemplo peticiones de compra, de perfil, etc.
 * Esta clase puede implementar cualquier tipo de petición cXML, aunque
 * tiene implementado constantes de un subconjunto de tipos de peticiones:
 * OrderRequest, ProfileRequest, SubscriptionListRequest,
```

```
 * SubscriptionContentRequest, SupplierListRequest, SupplierDataRequest.
 *
 *
 */
public class PeticionCXML extends DocumentoCXML {

    /**
     * Constante que representa el tipo de petición cXML "ProfileRequest"
     * (petición de perfil).
     */
    public final static String PETICION_PERFIL = "ProfileRequest";

    /**
     * Constante que representa el tipo de petición cXML "OrderRequest"
     * (petición de orden de compra).
     */
    public final static String PETICION_COMPRA = "OrderRequest";

    /**
     * Constante que representa el tipo de petición cXML
     * "SubscriptionListRequest" (petición de lista de catálogos).
     */
    public final static String PETICION_LISTA_CATALOGOS =
"SubscriptionListRequest";

    /**
     * Constante que representa el tipo de petición cXML
     * "SubscriptionContentRequest" (petición de contenido de catálogo).
     */
    public final static String PETICION_CATALOGO = "SubscriptionContentRequest";

    /**
     * Constante que representa el tipo de petición cXML
     * "SupplierListRequest" (petición de lista de proveedores).
     */
    public final static String PETICION_LISTA_PROVEEDORES =
"SupplierListRequest";

    /**
     * Constante que representa el tipo de petición cXML
     * "SupplierDataRequest" (petición de datos de proveedor).
     */
    public final static String PETICION_PROVEEDOR = "SupplierDataRequest";

    /**
     * Constante que representa un dominio de identidad de los credenciales
     * cXML por defecto.
     */
    public final static String DOMINIO_DE_IDENTIDAD = "UsuarioID";
```

```
/**
 * Constante que representa el agente software usado para enviar esta
 * petición cXML por defecto.
 */
protected final static String AGENTE = "Agente desconocido";

/**
 * Almacena si ya ha sido establecido un elemento de petición en el
 * elemento "Request".
 */
private boolean establecido = false;

/**
 * Constructor básico de la clase. Crea una nueva instancia de esta
 * clase creando una nueva estructura XML de petición cXML.
 */
public PeticionCXML() {
   super(TIPO_CXML);
   establecerEstructura();
}

/**
 * Crea una nueva instancia de una <code>PeticionCXML</code> del tipo
 * especificado.
 *
 */
public PeticionCXML(String tipo) {
   this();
   setTipoPeticion(tipo);
}

/**
 * Crea una nueva instancia de una <code>PeticionCXML</code> de tipo
 * y contenido el especificado por un <code>Element</code> JDOM.
 *
 */
public PeticionCXML(Element tipo) {
   this();
   setTipoPeticion(tipo);
}

/**
 * Devuelve un elemento JDOM con el tipo de petición cXML que representa
 * este objeto. El elemento JDOM devuelto incluye todos los nodos que
 * cuelgan del nodo tipo de petición.
 *
 */
public Element getTipoPeticion() {
   Element request = getElemento("cXML/Request");
   //Se obtiene una lista con todos los nodos que cuelgan del nodo Request
```

```
    List lista = request.getChildren();
    //Se escoge el primer nodo (aunque sólo debe de haber uno en la lista)
    return (Element) lista.get(0);
}

/**
 * Devuelve el tipo de petición cXML que representa este objeto.
 *
 */
public String getTipoPeticionEnString() {
    return getTipoPeticion().getName();
}

/**
 * Establece el tipo de petición cXML de este objeto. Este método sólo
 * añade una etiqueta XML con el nombre especificado por
 * <code>tipo</code>.
 *
 */
public void setTipoPeticion(String tipo) {
    Element peticion = getElemento("cXML/Request");
    Element tipoPeticion = new Element(tipo);

    peticion.addContent(tipoPeticion);
}

/**
 * Establece el tipo y contenido de la petición cXML que representa
 * este objeto mediante el <code>Element</code> especificado. Si
 * este método es llamado dos o más veces, sólo tendrá efecto la primera
 * vez que fue invocado, para evitar la adición de múltiples elementos
 * "Tipo"-Request al elemento "Request".
 *
 */
public void setTipoPeticion(Element tipo) {
  if ( !establecido ) {
      Element peticion = getElemento("cXML/Request");
      peticion.addContent(tipo);
      establecido = true;
    } else {
      /* Ya ha sido añadido un elemento, este no se añade */
    }
}

/**
 * Devuelve la identidad del elemento "From" de esta petición cXML.
 *
 */
public String getIdentidadFrom() {
    Element identidad = getElemento("cXML/Header/From/Credential/Identity");
```

```
    return identidad.getText();
}

/**
 * Establece la identidad del elemento "From" de esta petición cXML.
 *
 */
public void setIdentidadFrom(String id) {
    Element identidad = getElemento("cXML/Header/From/Credential/Identity");
    identidad.setText(id);
}

/**
 * Devuelve el valor del atributo "Domain" del elemento "Credential"
 * en el elemento "From" de esta petición cXML.
 *
 */
public String getDominioCredencialFrom() {
    Element credencial = getElemento("cXML/Header/From/Credential");
    return credencial.getAttributeValue("domain", (String) null);
}

/**
 * Establece el valor del atributo "Domain" del elemento "Credential"
 * en el elemento "From" de esta petición cXML.
 *
 */
public void setDominioCredencialFrom(String valor) {
    Element credencial = getElemento("cXML/Header/From/Credential");
    if (valor != null) {
        credencial.setAttribute("domain", valor);
    } else {
        //Se establece un valor por defecto
        credencial.setAttribute("domain", DOMINIO_DE_IDENTIDAD);
    }
}

/**
 * Devuelve la identidad del elemento "To" de esta petición cXML.
 *
 */
public String getIdentidadTo() {
    Element identidad = getElemento("cXML/Header/To/Credential/Identity");
    return identidad.getText();
}

/**
 * Establece la identidad del elemento "To" de esta petición cXML.
 *
 */
```

```
public void setIdentidadTo(String id) {
   Element identidad = getElemento("cXML/Header/To/Credential/Identity");
   identidad.setText(id);
}

/**
 * Devuelve el valor del atributo "Domain" del elemento "Credential"
 * en el elemento "To" de esta petición cXML.
 *
 */
public String getDominioCredencialTo() {
   Element credencial = getElemento("cXML/Header/To/Credential");
   return credencial.getAttributeValue("domain", (String) null);
}

/**
 * Establece el valor del atributo "Domain" del elemento "Credential"
 * en el elemento "To" de esta petición cXML.
 *
 */
public void setDominioCredencialTo(String valor) {
   Element credencial = getElemento("cXML/Header/To/Credential");
   if (valor != null) {
      credencial.setAttribute("domain", valor);
   } else {
      //Se establece un valor por defecto
      credencial.setAttribute("domain", DOMINIO_DE_IDENTIDAD);
   }
}

/**
 * Devuelve la identidad del elemento "Sender" de esta petición cXML.
 *
 */
public String getIdentidadSender() {
   Element identidad = getElemento("cXML/Header/Sender/Credential/Identity");
   return identidad.getText();
}

/**
 * Establece la identidad del elemento "Sender" de esta petición cXML.
 *
 */
public void setIdentidadSender(String id) {
   Element identidad = getElemento("cXML/Header/Sender/Credential/Identity");
   identidad.setText(id);
}

/**
 * Devuelve el valor del atributo "Domain" del elemento "Credential"
```

```
 * en el elemento "Sender" de esta petición cXML.
 *
 */
public String getDominioCredencialSender() {
   Element credencial = getElemento("cXML/Header/Sender/Credential");
   return credencial.getAttributeValue("domain", (String) null);
}

/**
 * Establece el valor del atributo "Domain" del elemento "Credential"
 * en el elemento "Sender" de esta petición cXML.
 *
 */
public void setDominioCredencialSender(String valor) {
   Element credencial = getElemento("cXML/Header/Sender/Credential");
   if (valor != null) {
      credencial.setAttribute("domain", valor);
   } else {
      //Se establece un valor por defecto
      credencial.setAttribute("domain", DOMINIO_DE_IDENTIDAD);
   }
}

/**
 * Devuelve el valor del atributo "SharedSecret" del elemento "Credential"
 * en el elemento "Sender" de esta petición cXML.
 *
 */
public String getClaveSender() {
   Element secreto = getElemento("cXML/Header/Sender/Credential/SharedSecret");
   return secreto.getText();
}

/**
 * Establece el valor del atributo "SharedSecret" del elemento "Credential"
 * en el elemento "Sender" de esta petición cXML.
 *
 */
public void setClaveSender(String clave) {
   Element secreto = getElemento("cXML/Header/Sender/Credential/SharedSecret");
   secreto.setText(clave);
}

/**
 * Devuelve el valor del atributo "UserAgent" del elemento "Sender"
 * de esta petición cXML.
 *
 */
public String getAgente() {
   Element agenteUsuario = getElemento("cXML/Header/Sender/UserAgent");
```

```
    return agenteUsuario.getText();
}

/**
 * Establece el valor del atributo "UserAgent" del elemento "Sender"
 * de esta petición cXML.
 *
 */
public void setAgente(String agente) {
    Element agenteUsuario = getElemento("cXML/Header/Sender/UserAgent");
    agenteUsuario.setText(agente);
}

/**
 * Crea el esqueleto básico de una petición cXML e inicializa algunos
 * valores por defecto usando JDOM.
 */
private void establecerEstructura() {
    Element raiz = getElementoRaiz();

    /*
     * Se construye el árbol JDOM de petición en 3 partes:
     * 1) Se asignan los atributos del elemento raíz cXML
     * 2) Se crea el elemento cabecera (header) del documento
     * 3) Se crea el elemento Request del documento
     */

    /*Los documentos de tipo cXML deben llevar los siguientes atributos:*/
    //Se genera un número aleatorio como identificación
    raiz.setAttribute("payloadID", FactoriaAleatoria.cadenaNumericaAleatoria());
    //Se establece el idioma del documento
    raiz.setAttribute("lang","es", Namespace.XML_NAMESPACE);
    //Se genera una fecha ISO8601
    raiz.setAttribute("timestamp", FechaISO8601.fechaISO() );

    /* Se crea la cabecera (Header) con los elementos From, To y Sender */
    Element cabecera = new Element("Header");
    Element from = new Element("From");
    Element to = new Element("To");
    Element sender = new Element("Sender");

    //Se construye el elemento From
    Element credencialFrom = new Element("Credential");
    Element identidadFrom = new Element("Identity");
    credencialFrom.setAttribute("domain", DOMINIO_DE_IDENTIDAD);
    credencialFrom.addContent(identidadFrom);
    from.addContent(credencialFrom);

    //Se construye el elemento To
    Element credencialTo = new Element("Credential");
```

```
        Element identidadTo = new Element("Identity");
        credencialTo.setAttribute("domain", DOMINIO_DE_IDENTIDAD);
        credencialTo.addContent(identidadTo);
        to.addContent(credencialTo);

        //Se construye el elemento Sender
        Element credencialSender = new Element("Credential");
        Element agente = new Element("UserAgent");
        Element identidadSender = new Element("Identity");
        Element secretoSender = new Element("SharedSecret");
        credencialSender.setAttribute("domain", DOMINIO_DE_IDENTIDAD);
        credencialSender.addContent(identidadSender);
        credencialSender.addContent(secretoSender);
        agente.setText(AGENTE);
        sender.addContent(credencialSender);
        sender.addContent(agente);

        //Se añaden al elemento Header (cabecera)
        cabecera.addContent(from);
        cabecera.addContent(to);
        cabecera.addContent(sender);

        /* Se construye el elemento petición (Request) */
        Element peticion = new Element("Request");

        /* Se añaden los elementos cabecera y petición al elemento raiz CXML */
        raiz.addContent(cabecera);
        raiz.addContent(peticion);
    }

}
```

1.2.7. Clase PeticionDeCompraCXML

```
//
//   Fichero: PeticionDeCompraCXML.java
//   Autor: Trinitario Meseguer Calero
//   Fecha: 21/02/05
//   Descripción: Se trata de una subclase de la clase PeticionCXML, por
//      lo que poseerá todos sus métodos. Así como con la clase
//      PeticionCXML establece la cabecera de la petición, que es
//      genérica para todas las peticiones en cXML, esta clase establece
//      el cuerpo específico de la petición de compra, OrderRequest. Posee
//      también los métodos necesarios para completar todos los campos de
//      la petición de compra. La información para completar la petición de
//      compra se extrae por una parte de los datos introducidos por el
//      usuario en el panel de “Datos del Comprador”, que es mostrado por la
//      clase DatosComprador, y por otra del propio fichero XML del
//      producto que se desea comprar.
//
```

```
package PFC.codigo.xml;

import PFC.codigo.acciones.*;
import PFC.codigo.jxta.*;
import PFC.codigo.tablas.*;
import PFC.codigo.utiles.*;

import org.jdom.*;

public class PeticionDeCompraCXML extends PeticionCXML {

   /** Creates a new instance of PeticionDeCompraCXML */
   public PeticionDeCompraCXML() {
      super();
      estableceCuerpoPeticion();

   }

//   Establece el nombre de la entidad a la que se le facturará
   void setNombreFacturacion(String nombre){
      Element name =
getElemento("cXML/Request/OrderRequest/OrderRequestHeader/BillTo/Address/Nam
e");
      name.setText(nombre);

   }

//   Establece la dirección de facturación del documento petición de compra
   void setDireccionFacturacion(String direccion){
      Element address =
getElemento("cXML/Request/OrderRequest/OrderRequestHeader/BillTo/Address/Posta
lAddress/Street");
      address.setText(direccion);

   }

//   Establece el código postal de facturación del documento petición de compra
   void setCPFacturacion(String cp){
      Element postalCode =
getElemento("cXML/Request/OrderRequest/OrderRequestHeader/BillTo/Address/Posta
lAddress/PostalCode");
      postalCode.setText(cp);

   }

//   Establece la localidad de facturación del documento petición de compra
   void setLocalidadFacturacion(String localidad){
      Element city =
getElemento("cXML/Request/OrderRequest/OrderRequestHeader/BillTo/Address/Posta
```

```
lAddress/City");
        city.setText(localidad);
    }

//    Establece la provincia de facturación del documento petición de compra
    void setProvinciaFacturacion(String provincia){
        Element state =
getElemento("cXML/Request/OrderRequest/OrderRequestHeader/BillTo/Address/Posta
lAddress/State");
        state.setText(provincia);
    }

//    Establece el país de facturación del documento petición de compra
    void setPaisFacturacion(String pais){
        Element country =
getElemento("cXML/Request/OrderRequest/OrderRequestHeader/BillTo/Address/Posta
lAddress/Country");
        country.setText(pais);
    }

//    Establece el nombre de la persona o entidad de envío en el documento
//    petición de compra
    void setNombreEnvio(String nombre){
        Element name =
getElemento("cXML/Request/OrderRequest/OrderRequestHeader/ShipTo/Address/Post
alAddress/DeliverTo");
        name.setText(nombre);
    }

//    Establece la dirección de envío del documento petición de compra
    void setDireccionEnvio(String direccion){
        Element address =
getElemento("cXML/Request/OrderRequest/OrderRequestHeader/ShipTo/Address/Post
alAddress/Street");
        address.setText(direccion);
    }

//    Establece la código postal de envío del documento petición de compra
    void setCPEnvio(String cp){
        Element postalCode =
getElemento("cXML/Request/OrderRequest/OrderRequestHeader/ShipTo/Address/Post
alAddress/PostalCode");
        postalCode.setText(cp);
    }

//    Establece la localidad de envío del documento petición de compra
    void setLocalidadEnvio(String localidad){
        Element city =
getElemento("cXML/Request/OrderRequest/OrderRequestHeader/ShipTo/Address/Post
alAddress/City");
```

```
        city.setText(localidad);
    }

//    Establece la provincia de envío del documento petición de compra
    void setProvinciaEnvio(String provincia){
        Element state =
getElemento("cXML/Request/OrderRequest/OrderRequestHeader/ShipTo/Address/Post
alAddress/State");
        state.setText(provincia);
    }

//    Establece el país de envío del documento petición de compra
    void setPaisEnvio(String pais){
         Element country =
getElemento("cXML/Request/OrderRequest/OrderRequestHeader/ShipTo/Address/Post
alAddress/Country");
        country.setText(pais);
    }

//    Establece la fecha de caducidad de la tarjeta de crédito del documento
//    petición de compra
    void setFechaCaducidad(String fecha){
        Element pcard =
getElemento("cXML/Request/OrderRequest/OrderRequestHeader/Payment/PCard");
        pcard.setAttribute("expiration",fecha);
    }

//    Establece el número de la tarjeta de crédito del documento petición
//    de compra
    void setNumeroTarjeta(String numeroTarjeta){
        Element pcard =
getElemento("cXML/Request/OrderRequest/OrderRequestHeader/Payment/PCard");
        pcard.setAttribute("number",numeroTarjeta);

    }

//    Rellena con los datos del producto el documento de petición de compra
    void setDatosProducto(ProductoXML producto){

        Element credential = getElemento("cXML/Header/To/Credential");
        credential.setAttribute("domain",producto.dominio.getText());

        Element identity = getElemento("cXML/Header/To/Credential/Identity");
        identity.setText(producto.identidad.getText());

        Element money =
getElemento("cXML/Request/OrderRequest/OrderRequestHeader/Total/Money");
        money.setText(producto.precio.getText());

        Element moneyItem =
```

```
getElemento("cXML/Request/OrderRequest/ItemOut/ItemDetail/UnitPrice/Money");
        moneyItem.setText(producto.precio.getText());

        Element supplierPartID =
getElemento("cXML/Request/OrderRequest/ItemOut/ItemID/SupplierPartID");
        supplierPartID.setText(producto.codigoProd.getText());

        Element description =
getElemento("cXML/Request/OrderRequest/ItemOut/ItemDetail/Description");
        description.setText(producto.nombreProducto.getText());

        Element clasification =
getElemento("cXML/Request/OrderRequest/ItemOut/ItemDetail/Classification");
        clasification.setAttribute("domain",producto.categoria.getText());
        clasification.setText(producto.subcategoria.getText());

    }

//    Establece el cuerpo de un documento de petición de compra en el
//    protocolo cXML
    void estableceCuerpoPeticion(){

        Element orderRequest=new Element("OrderRequest");
        Element orderRequestHeader=new Element("OrderRequestHeader");
        Element itemOut=new Element("ItemOut");

        //Se genera un número aleatorio como identificación
        orderRequestHeader.setAttribute("orderID",
FactoriaAleatoria.cadenaNumericaAleatoria());

        //Se genera una fecha ISO8601
        orderRequestHeader.setAttribute("orderDate", FechaISO8601.fechaISO() );

        orderRequest.addContent(orderRequestHeader);
        orderRequest.addContent(itemOut);

        itemOut.setAttribute("quantity","1");


//        Se construye el elemento OrderRequestHeader

        Element total=new Element("Total");
        Element shipTo=new Element("ShipTo");
        Element billTo=new Element("BillTo");
        Element payment=new Element("Payment");


        orderRequestHeader.addContent(total);
        orderRequestHeader.addContent(shipTo);
        orderRequestHeader.addContent(billTo);
```

```
        orderRequestHeader.addContent(payment);

//      Se construye elemento Total

        Element money=new Element("Money");
        money.setAttribute("currency","EUR");

        total.addContent(money);

//      Se construye el elemento ShipTo
        Element nameEnv=new Element("Name");
        Element addressEnv=new Element("Address");
        Element postalAddressEnv=new Element("PostalAddress");
        Element deliverToEnv=new Element("DeliverTo");
        Element streetEnv=new Element("Street");
          Element cityEnv=new Element("City");
        Element stateEnv=new Element("State");
        Element postalCodeEnv=new Element("PostalCode");
        Element countryEnv=new Element("Country");

        countryEnv.setAttribute("isoCountryCode","ES");
        nameEnv.setAttribute("lang","es",Namespace.XML_NAMESPACE);

        shipTo.addContent(addressEnv);
        addressEnv.addContent(nameEnv);
        addressEnv.addContent(postalAddressEnv);
        postalAddressEnv.addContent(deliverToEnv);
        postalAddressEnv.addContent(streetEnv);
        postalAddressEnv.addContent(cityEnv);
        postalAddressEnv.addContent(stateEnv);
        postalAddressEnv.addContent(postalCodeEnv);
        postalAddressEnv.addContent(countryEnv);

//      Se construye el elemento BillTo

        Element addressFact=new Element("Address");
        Element nameFact=new Element("Name");
        Element postalAddressFact=new Element("PostalAddress");
        Element deliverToFact=new Element("DeliverTo");
        Element streetFact=new Element("Street");
          Element cityFact=new Element("City");
        Element stateFact=new Element("State");
        Element postalCodeFact=new Element("PostalCode");
        Element countryFact=new Element("Country");

        countryFact.setAttribute("isoCountryCode","ES");
        nameFact.setAttribute("lang","es",Namespace.XML_NAMESPACE);

        billTo.addContent(addressFact);
        addressFact.addContent(nameFact);
```

```
        addressFact.addContent(postalAddressFact);
        postalAddressFact.addContent(deliverToFact);
        postalAddressFact.addContent(streetFact);
        postalAddressFact.addContent(cityFact);
        postalAddressFact.addContent(stateFact);
        postalAddressFact.addContent(postalCodeFact);
        postalAddressFact.addContent(countryFact);

//      Se construye el elemento Payment
        Element pCard=new Element("PCard");

        payment.addContent(pCard);

//      Se construye el elemento ItemOut
        Element itemID=new Element("ItemID");
        Element itemDetail=new Element("ItemDetail");

        itemOut.addContent(itemID);
        itemOut.addContent(itemDetail);

         Element supplierPartID=new Element("SupplierPartID");

         itemID.addContent(supplierPartID);

         Element unitPrice=new Element("UnitPrice");
         Element moneyItem=new Element("Money");
         Element description=new Element("Description");
         Element unitOfMeasure=new Element("UnitOfMeasure");
         Element classification=new Element("Classification");

         moneyItem.setAttribute("currency","EUR");

         unitPrice.addContent(moneyItem);

         itemDetail.addContent(unitPrice);
         itemDetail.addContent(description);
         itemDetail.addContent(unitOfMeasure);
         itemDetail.addContent(classification);

         description.setAttribute("lang","es",Namespace.XML_NAMESPACE);
         classification.setAttribute("domain","");


//      Se añade todo el cuerpo de la petición al documento cXML
        setTipoPeticion(orderRequest);

    }
}
```

1.2.8. Clase PeticionDePerfilCXML

```
//
//   Fichero: PeticionDePerfilCXML.java
//   Autor: Trinitario Meseguer Calero
//   Fecha: 07/03/05
//   Descripción: Esta clase es subclase de PeticionCXML y se encarga
//      de establecer el cuerpo específico de una petición de perfil
//      en cXML, ProfileRequest. En este caso el cuerpo de la petición no
//      tiene ningún dato que haya que introducir por parte del usuario,
//      no así la cabecera que debe tomar los datos al igual que la petición
//      de compra del panel de “Datos del Comprador” y del fichero XML
//      del producto a comprar.
//

package PFC.codigo.xml;

import PFC.codigo.acciones.*;
import PFC.codigo.jxta.*;
import PFC.codigo.tablas.*;
import PFC.codigo.utiles.*;

import org.jdom.*;

public class PeticionDePerfilCXML extends PeticionCXML{


   /** Creates a new instance of PeticionDePerfilCXML */
   public PeticionDePerfilCXML() {
      super();
      estableceCuerpoPeticion();

   }

//   Método que establece el cuerpo de una petición de perfil
//   del protocolo cXML
   public void estableceCuerpoPeticion(){
      Element profileRequest=new Element("ProfileRequest");
      setTipoPeticion(profileRequest);


   }
}
```

1.2.9. Clase RespuestaCXML

```
//
//   Fichero: RespuestaCXML.java
//   Autor: Trinitario Meseguer Calero
//   Fecha: 15/01/05
//   Descripción: Contiene la clase RespuestaCXML, que representa a un documento
```

```
//          de contenido cXML de respuesta.
//


package PFC.codigo.xml;

import PFC.codigo.acciones.*;
import PFC.codigo.jxta.*;
import PFC.codigo.tablas.*;
import PFC.codigo.utiles.*;

import java.util.List;
import org.jdom.*;



/**
 * Clase que representa un documento de respuesta basado en cXML. <br>
 * En el modelo de intercambio de mensajes de comercio eléctronico
 * propuesto por cXML, los concentradores o servidores de comercio
 * eléctronico envían respuestas cXML ante las peticiones de los clientes.
 * Esta clase puede implementar cualquier tipo de respuesta cXML, aunque
 * tiene implementado constantes de un subconjunto de tipos de peticiones:
 * OrderResponse, ProfileResponse, SubscriptionListResponse,
 * SubscriptionContentResponse, SupplierListResponse, SupplierDataResponse.
 *
 *
 */
public class RespuestaCXML extends DocumentoCXML {

   /**
    * Constante que representa el tipo de respuesta cXML "ProfileResponse"
    * (respuesta de perfil).
    */
   public final static String RESPUESTA_PERFIL = "ProfileResponse";

   /**
    * Constante que representa el tipo de respuesta cXML "OrderResponse"
    * (respuesta de orden de compra).
    */
   public final static String RESPUESTA_COMPRA = "OrderResponse";

   /**
    * Constante que representa el tipo de respuesta cXML
    * "SubscriptionListResponse" (respuesta de lista de catálogos).
    */
   public final static String RESPUESTA_LISTA_CATALOGOS =
"SubscriptionListResponse";

   /**
```

```
    * Constante que representa el tipo de respuesta cXML
    * "SubscriptionContentResponse" (respuesta de contenido de catálogo).
    */
   public final static String RESPUESTA_CATALOGO =
"SubscriptionContentResponse";

   /**
    * Constante que representa el tipo de respuesta cXML
    * "SupplierListResponse" (respuesta de lista de proveedores).
    */
   public final static String RESPUESTA_LISTA_PROVEEDORES =
"SupplierListResponse";

   /**
    * Constante que representa el tipo de respuesta cXML
    * "SupplierDataResponse" (respuesta de datos de proveedor).
    */
   public final static String RESPUESTA_PROVEEDOR = "SupplierDataResponse";

   /**
    * Almacena si ya ha sido establecido un elemento de respuesta en el
    * elemento "Response".
    */
   private boolean establecido = false;

   /**
    * Constructor básico de la clase. Crea una nueva instancia de esta
    * clase creando una nueva estructura XML de respuesta cXML.
    */
   public RespuestaCXML() {
      super(TIPO_CXML);
      establecerEstructura();
   }

   /**
    * Crea una nueva instancia de una <code>RespuestaCXML</code> e
    * inicializa el elemento "Status" de la respuesta cXML al valor
    * especificado. Este método está especialmente implementado para
    * crear una nueva OrderResponse, puesto que éste tipo de documento
    * cXML sólo requiere como respuesta el elemento Status.
    *
    */
   public RespuestaCXML(String codigo, String texto, String descr) {
      this();
      setStatus(codigo, texto, descr);
   }

   /**
    * Devuelve un elemento JDOM con el tipo de respuesta cXML que representa
    * este objeto. El elemento JDOM devuelto incluye todos los nodos que
```

```
 * cuelgan del nodo tipo de respuesta
 *
 */
public Element getTipoRespuesta() {
   Element respuesta = getElemento("cXML/Response");
   //Se obtiene una lista con todos los nodos que cuelgan del nodo Response
   List lista = respuesta.getChildren();

   /*
    * Un elemento Response debe tener obligatoriamente un elemento
    * Status, y puede tener o no un elemento "Tipo"-Response. No lo tendrá
    * si es una OrderResponse. Se comprueba que la lista de elementos
    * colgantes de Response tenga al menos 2 elementos; si no, es una
    * OrderResponse y se devuelve null.
    */
   return ( (lista.size() > 1) ? ((Element) lista.get(1))
                               : null );
}

/**
 * Devuelve el tipo de respuesta cXML que representa este objeto.
 *
 */
public String getTipoRespuestaEnString() {
   Element respuesta = getTipoRespuesta();
   /*
    * Si el elemento Response no contiene ningún tipo de respuesta,
    * la respuesta es de tipo OrderRequest. En el otro caso, se devuelve
    * el tipo de la respuesta cXML.
    */
   return ( (respuesta == null) ? RESPUESTA_COMPRA
                                : respuesta.getName() );
}

/**
 * Establece el tipo y contenido de la respuesta cXML que representa
 * este objeto mediante el <code>Element</code> especificado. Si
 * este método es llamado dos o más veces, sólo tendrá efecto la primera
 * vez que fue invocado, para evitar la adición de múltiples elementos
 * "Tipo"-Response al elemento "Response".
 *
 */
public void setTipoRespuesta(Element tipo) {
   if ( !establecido ) {
      Element respuesta = getElemento("cXML/Response");
      respuesta.addContent(tipo);
      establecido = true;
   } else {
      /* Ya ha sido añadido un elemento, este no se añade */
   }
```

```
}

/**
 * Devuelve el valor del atributo "code" del elemento "Status" de
 * esta <code>RespuestaCXML</code>.
 *
 */
public String getCodigoStatus() {
   Element status = getElemento("cXML/Response/Status");
   return status.getAttributeValue("code");
}

/**
 * Devuelve el valor del atributo "text" del elemento "Status" de
 * esta <code>RespuestaCXML</code>, que representa un valor canónico
 * de código HTTP.
 *
 */
public String getTextoStatus() {
   Element status = getElemento("cXML/Response/Status");
   return status.getAttributeValue("text");
}


/**
 * Devuelve el valor del elemento "Status" de esta
 * <code>RespuestaCXML</code>, que es una explicación específica
 * del error ocurrido.
 *
 */
public String getDescripcionStatus() {
   return getElemento("cXML/Response/Status").getText();
}


/**
 * Establece el valor de los atributos "code" y "text" del elemento
 * "Status" de esta <code>RespuestaCXML</code>.
 *
 */
public void setStatus(String codigo, String texto, String descr) {
   Element status = getElemento("cXML/Response/Status");
   status.setAttribute("code", codigo);
   status.setAttribute("text", texto);
   status.setText(descr);
}

/**
 * Crea el esqueleto básico de una respuesta cXML e inicializa algunos
 * valores por defecto usando JDOM.
```

```
    */
   protected void establecerEstructura() {
      Element raiz = getElementoRaiz();

      /*
       * Se construye el árbol JDOM de respuesta en 2 partes:
       * 1) Se asignan los atributos del elemento raíz cXML
       * 2) Se crea el elemento Response del documento
       */

      /*Los documentos de tipo cXML deben llevar los siguientes atributos:*/
      //Se genera un número aleatorio como identificación
      raiz.setAttribute("payloadID", FactoriaAleatoria.cadenaNumericaAleatoria());
      //Se establece el idioma del documento
      raiz.setAttribute("lang","es", Namespace.XML_NAMESPACE);
      //Se genera una fecha ISO8601
      raiz.setAttribute("timestamp", FechaISO8601.fechaISO() );

      /* Se construye el elemento Response */
      Element respuesta = new Element("Response");
      //Se construye el elemento status y se añade al elemento Response
      Element status = new Element("Status");
      status.setAttribute("code","");
      status.setAttribute("text","");
      respuesta.addContent(status);

      /*Se añade el elemento Response al elemento CXML */
      raiz.addContent(respuesta);
   }
}
```

1.3. PAQUETE PFC.CODIGO.JXTA

1.3.1. Clase PlataformaJXTA

```
//
//   Fichero: PlataformaJXTA.java
//   Autor: Trinitario Meseguer Calero
//   Fecha: 03/12/04
//   Descripción: La principal función de esta clase es la conexión a la
//      red JXTA y dentro de esta al grupo de peers especifico al que
//      se conectaran todos los usuarios de “Comercio Electrónico 1.0”,
//      donde podrán anunciar y buscar aquellos  productos que les interese.
//

package PFC.codigo.jxta;

import PFC.codigo.acciones.*;
import PFC.codigo.tablas.*;
```

```
import PFC.codigo.utiles.*;
import PFC.codigo.xml.*;

import java.io.FileInputStream;
import java.io.StringWriter;
import java.util.*;

import net.jxta.rendezvous.RendezVousService;
import net.jxta.document.AdvertisementFactory;
import net.jxta.document.Element;
import net.jxta.document.MimeMediaType;
import net.jxta.document.StructuredDocument;
import net.jxta.document.StructuredDocumentFactory;
import net.jxta.document.StructuredDocumentUtils;
import net.jxta.document.StructuredTextDocument;
import net.jxta.endpoint.Message;
import net.jxta.endpoint.MessageElement;
import net.jxta.exception.PeerGroupException;
import net.jxta.id.ID;
import net.jxta.id.IDFactory;
import net.jxta.peergroup.PeerGroup;
import net.jxta.peergroup.PeerGroupFactory;
import net.jxta.pipe.InputPipe;
import net.jxta.pipe.PipeService;
import net.jxta.platform.ModuleClassID;
import net.jxta.protocol.ModuleClassAdvertisement;
import net.jxta.protocol.ModuleSpecAdvertisement;
import net.jxta.protocol.PeerGroupAdvertisement;
import net.jxta.protocol.PipeAdvertisement;
import net.jxta.discovery.*;
import net.jxta.peer.*;
import net.jxta.credential.Credential;
import net.jxta.membership.Authenticator;
import net.jxta.credential.AuthenticationCredential;
import net.jxta.membership.MembershipService;
import net.jxta.protocol.ModuleImplAdvertisement;
import java.io.IOException;


public class PlataformaJXTA {

   public PeerGroup group;
   public PeerGroupAdvertisement groupAdvertisement;
   public DiscoveryService discovery;
   public PipeService pipes;

   protected Vector misGrupos;
   private int intentos=10;
   private int tiempo=6*1000;
```

```
   /** Creates a new instance of PlataformaJXTA */
   public PlataformaJXTA() {
      misGrupos = new Vector(20);
   }

//    Método básico para inicializar la red JXTA, se instancia el grupo por
//    defecto y se obtiene el anuncio de grupo
   public void startJXTA() {
      try {
         // Se instancia el grupo por defecto, NetPeerGroup
         group = PeerGroupFactory.newNetPeerGroup();
         misGrupos.add(group);

      } catch (PeerGroupException e) {
         // No se puede instanciar el grupo y se sale de la aplicación
         System.out.println("fatal error : group creation failure");
         e.printStackTrace();
         System.exit(1);
      }
//       Se obtiene el anuncio de grupo
      groupAdvertisement = group.getPeerGroupAdvertisement();

      System.out.println("Getting DiscoveryService");
      discovery =group.getDiscoveryService();
      System.out.println("Getting PipeService");
      pipes =group.getPipeService();



}

    public void limpiarCache() {

      /*
       * En los tutoriales de JXTA y en los JAVADOC se usa el método
       * flushAdvertisements() para borrar el caché. Sin embargo, este
       * método no funciona en absoluto. Este error está registrado
       * como "Platform issue 344" en los archivos de bugs de JXTA.
       * En la versión de JXTA usada aquí, pues, no se puede limpiar el
       * caché. Por esto se propone una forma alternativa y válida de
       * realizarlo, mediante un borrado del directorio donde se almacenan
       * los resultados. De todas maneras, se incluye el código que
       * es erróneo debido a error de JXTA.
       */

      /*
       *Esta es la forma propuesta para el borrado de caché:
       */
      UtilesDirectorio.borrarDirectorio(".jxta/cm");
```

```
  }

    public boolean conectarConRendezvous(int intentos, int tiempo) {
      boolean resultado = false;
      //Se obtiene el servicio JXTA de rendezvous
      RendezVousService servicio = group.getRendezVousService();

      /*
       * Se itera mientras que no esté conectado al rendezvous o queden
       * más intentos restantes.
       */
      for( int i=0; (i < intentos) && (resultado == false) ; i++) {
         if( servicio.isConnectedToRendezVous() == true ) {
            //Ya está conectado
            resultado = true;
         } else {
            try {
               Thread.sleep(tiempo*1000);
            } catch (InterruptedException ie) {   }
         }
      }
      return resultado;
  }
    public boolean estaConectadoConRendezvous() {
      RendezVousService servicio = group.getRendezVousService();
      return servicio.isConnectedToRendezVous();
  }
    /**
   * Realiza los trámites para entrar a forma parte de un grupo de peers JXTA
   * especificado a partir de su nombre. Si el grupo no existe, este método
   * lo crea a partir del nombre y la descripción especificados y se une
   * a él. Devuelve una instancia del grupo al que se ha entrado.
   * <b>Nota:</b>
   * <i>Este método puede tardar tiempo en completarse, puesto que realiza
   * una búsqueda de grupos por la red JXTA. </i>
   *
   */
  public PeerGroup entrarEnGrupo(String nombre,
                                 String descripcion)
                          {
      PeerGroupAdvertisement anuncioGrupo = null;

      //Se realiza una búsqueda por la red JXTA y por el caché después.
      descubrir(null, DiscoveryService.GROUP, "Name", nombre, null);
      Enumeration enumGrupos = descubrirDelCache(DiscoveryService.GROUP,
"Name", nombre);
      if ( enumGrupos.hasMoreElements() == true ) {
         //Está ya en el caché
         anuncioGrupo = (PeerGroupAdvertisement) enumGrupos.nextElement();
```

```
      } else {
         /*
          * No está en el caché, por lo que no existe el grupo. Se crea
          * y publica por tanto el grupo.
          */
         anuncioGrupo = crearNuevoGrupo(nombre, descripcion);
      }

      /* Se une al grupo */
      if ( entrarEnGrupo( anuncioGrupo ) == false ) {
         System.out.println("No se ha podido entrar en el grupo"
                           + nombre);
      }
      //Se obtiene la instancia del grupo
      group = obtenerGrupoUnido(anuncioGrupo);

      /*
       * Se devuelve un nuevo objeto GrupoDePeers del grupo al que se ha
       * adherido este objeto.
       */
      return (group);
   }

//   Método que intenta unirse al grupo de peers cuyo anuncio se pasa
//   como argumento
   public boolean entrarEnGrupo(PeerGroupAdvertisement anuncioGrupo) {
      boolean resultado;

      //Primero se comprueba que no esté ya dentro del grupo
      if ( obtenerGrupoUnido(anuncioGrupo) == null ) {
         PeerGroup miNuevoGrupo = null;
         try {
            //Se instancia el grupo al que se quiere unir
            miNuevoGrupo = group.newGroup(anuncioGrupo);

            /* Se crean los credenciales para unirse al grupo */
            StructuredDocument credencial = null;
            AuthenticationCredential credencialAutenticacion =
                        new AuthenticationCredential(miNuevoGrupo,
                                                     null,
                                                     credencial);
            //Para la adhesión, hay que usar el servicio de membresía JXTA
            MembershipService servicioMembresia =
                              miNuevoGrupo.getMembershipService();
            Authenticator autenticador =
                     servicioMembresia.apply(credencialAutenticacion);
            //Comienza la operación de adhesión al grupo
            if ( autenticador.isReadyForJoin() == true ) {
               //Se une al grupo
               Credential miCredencial =
```

```
                        servicioMembresia.join(autenticador);
                //Se añade a la lista interna de grupos ya unidos
                misGrupos.add(miNuevoGrupo);
                //Si llega aquí, es que ha tenido éxito.
                System.out.println("Unidos al
grupo:"+miNuevoGrupo.getPeerGroupName());
                resultado = true;
             } else {
                //No ha podido unirse al grupo
                resultado = false;
             }
          } catch (Exception e) {
             resultado = false;
          }
       } else {
          //Ya estaba unido al grupo
          resultado = true;
       }
       return resultado;
    }


/**
     * Realiza una búsqueda de anuncios de un tipo especificado en la red
     * JXTA, limitándola a anuncios con el valor especificado por
     * <code>valor</code> de un atributo especificado <code>atributo</code>.
     * Se puede también limitar la búsqueda a un sólo peer.
     * El parámetro <code>tipo</code> está referenciado en constantes de la
     * clase JXTA <code>DiscoveryService</code>.
     *
     */
    protected void descubrir(String peerID,
                             int tipo,
                             String atributo,
                             String valor,
                             DiscoveryListener discoListener) {
       //Se crea un objeto de servicio de descubrimiento
       DiscoveryService discoSvc = group.getDiscoveryService();
       /*
        * Se realizan varias búsquedas de anuncios, para asegurarse de que
        * se da tiempo a los otros peers de responder.
        */
       for (int i=0; i < intentos ; i++) {
          //Se inicia la búsqueda
          discoSvc.getRemoteAdvertisements(peerID,
                                           tipo,
                                           atributo,
                                           valor,
                                           10,
                                           discoListener);
```

```
        //Esperamos el intervalo de tiempo entre intentos
        try {
           Thread.sleep(tiempo);
        } catch(InterruptedException ie) {  }
     }
  }

  /**
   * Realiza una búsqueda de anuncios de un tipo especificado en el caché
   * JXTA, limitándola a anuncios con el valor especificado por
   * <code>valor</code> de un atributo especificado <code>atributo</code>.
   * El parámetro <code>tipo</code> está referenciado en constantes de la
   * clase JXTA <code>DiscoveryService</code>.
   *
   */
  protected Enumeration descubrirDelCache(int tipo,
                                  String atributo,
                                  String valor) {
     Enumeration enum = null;
     //Se crea un objeto de servicio de descubrimiento
     DiscoveryService disco = group.getDiscoveryService();

     try {
        //Realizamos un búsqueda en el caché
        enum = disco.getLocalAdvertisements(tipo, atributo, valor);
     } catch (IOException ioe) {
        //Ha ocurrido un error al mirar el caché.
        enum = null;
     }
     return enum;
  }
  /**
   * Comprueba que el este objeto ya pertenezca al grupo especificado
   * por <code>anuncioGrupo</code> y devuelve la instancia
   * <code>PeerGroup</code> de ese grupo.
   *
   */
  public PeerGroup obtenerGrupoUnido(PeerGroupAdvertisement anuncioGrupo) {

     PeerGroup resultado = null;
     Enumeration grupos = misGrupos.elements();

     //Se recorre la lista hasta que se acabe o se encuentre el grupo buscado
     while ( ( grupos.hasMoreElements() == true ) && (resultado == null) ) {
        PeerGroup aux = (PeerGroup) grupos.nextElement();
        //Se comprueba si es el buscado por el ID de grupo JXTA
        if ( aux.getPeerGroupID().equals(anuncioGrupo.getPeerGroupID()) ) {
           //Encontrado, ya no se busca más
           resultado = aux;
        }
```

```
      }

      return resultado;
   }
   /**
    * Crea un nuevo grupo de peers dados un nombre de grupo y una descripción
    * determinados.
    *
    */
   public PeerGroupAdvertisement crearNuevoGrupo(String nombre,
                                    String descripcion) {
      PeerGroup miNuevoGrupo;
      PeerGroupAdvertisement anuncioNuevoGrupo = null;

      try {
         /* Se crea el nuevo grupo */
         ModuleImplAdvertisement anuncioImpl =
                group.getAllPurposePeerGroupImplAdvertisement();
         miNuevoGrupo = group.newGroup(null,
                                anuncioImpl,
                                nombre,
                                descripcion);
         anuncioNuevoGrupo = miNuevoGrupo.getPeerGroupAdvertisement();

         /* Se publican los anuncios del nuevo grupo */
         DiscoveryService discoRaiz = group.getDiscoveryService();
         //Se publica localmente
         discoRaiz.publish(anuncioNuevoGrupo, DiscoveryService.GROUP);
         //Se publica al exterior
         discoRaiz.remotePublish(anuncioNuevoGrupo, DiscoveryService.GROUP);
      } catch (Exception e) {
         //Si ocurre algún error, no se ha creado nada
         anuncioNuevoGrupo = null;
      }
      return anuncioNuevoGrupo;
   }
}
```

1.3.2. Clase ServidorServiciosJXTA

```
//
//   Fichero: ServidorServiciosJXTA.java
//   Autor: Trinitario Meseguer Calero
//   Fecha: 20/01/05
//   Descripción: Todo "peer" tiene dos partes, una parte cliente y otra
//      servidor. Esta clase trata de implementar la parte servidor
//      de todo peer.Cada vez que se anuncia un nuevo producto,
//      esta clase es la encargada de crear un nuevo servicio, cuyo
//      nombre es el nombre del producto que estamos anunciando,
//      y publica su anuncio en la red JXTA. Por cada producto anunciado
```

```
//      se crea un nuevo hilo que estará esperando de forma indefinida
//      mientras esté en ejecución el programa o hasta que dicho producto
//      se venda. Esta clase está a la espera de que un cliente solicite
//      nuestro servicio anunciado, en este momento se crea una conexión
//      bidireccional entre el cliente y el servidor a través de la cual
//      ambos se intercambian mensajes JXTA que encapsularan a los
//      mensajes cXML.
//


package PFC.codigo.jxta;

import PFC.codigo.acciones.*;
import PFC.codigo.tablas.*;
import PFC.codigo.utiles.*;
import PFC.codigo.xml.*;

import java.net.*;
import net.jxta.pipe.*;
import net.jxta.discovery.*;
import net.jxta.protocol.*;
import net.jxta.document.*;
import net.jxta.endpoint.*;
import net.jxta.platform.*;
import net.jxta.util.*;
import net.jxta.id.*;
import net.jxta.peergroup.*;
import java.lang.*;
import org.apache.log4j.*;
import java.io.*;
import java.util.*;



public class ServidorServiciosJXTA  implements PipeMsgListener, Runnable  {

   String
rutaDirectorioPeticionesDeCompraRecibidas="./PeticionesDeCompraRecibidas/";
    String rutaDirectorioProductosAnunciados="./Anuncios/";
   Vender venta=null;


   private PipeAdvertisement pipeadv;
   private JxtaServerPipe serverPipe;
   private JxtaBiDiPipe bipipe;

   public PeerGroup group;
   public DiscoveryService discovery;
   private final static String SenderMessage = "ComercioP2P";
   String nombreFichero;
```

```
   private Thread hilo;
   private final static Logger LOG =
Logger.getLogger(ServidorServiciosJXTA.class.getName());
   ModeloTablaAnunciados modeloAnunciados;
   ModeloTablaVendidos modeloVendidos;
   int filaAnunciados;
   ProductoXML producto;

   Fichero fichero=new Fichero();
   ModuleSpecAdvertisement mdadv;

   /** Creates a new instance of ServidorServiciosJXTA */
   public ServidorServiciosJXTA(PlataformaJXTA plataforma, String nombreFichero,
ModeloTablaAnunciados modeloAnunciados, ModeloTablaVendidos modeloVendidos)
{

      group=plataforma.group;
      discovery=group.getDiscoveryService();
      this.nombreFichero=nombreFichero;
      hilo=new Thread(this);
      this.modeloAnunciados=modeloAnunciados;
      this.modeloVendidos=modeloVendidos;
      try{
         producto=new ProductoXML(new
File(rutaDirectorioProductosAnunciados+nombreFichero));
      }
      catch (Exception e){

         System.out.println("Excepcion al leer fichero anunciado"+e);
      }

   }

//   Método llamado cada vez que se recibe un mensaje por la tubería
//   bidireccional por parte del cliente. En función del mensaje recibido
//   que puede ser una petición de perfil o una petición de compra, se
//   enviará al cliente una respuesta de perfil o una respuesta de compra
   public void pipeMsgEvent(net.jxta.pipe.PipeMsgEvent event) {
       MessageElement tipoMensaje;
       Message msg = null;
       String rutaFichero;
      try {
         // grab the message from the event
         msg = event.getMessage();
         if (msg == null) {
            if (LOG.isEnabledFor(Level.DEBUG)) {
               LOG.debug("Received an empty message, returning");
            }
            return;
         }
```

```
        if (LOG.isEnabledFor(Level.DEBUG)) {
           LOG.debug("Received a response");
        }
        // get the message element named SenderMessage
          tipoMensaje = msg.getMessageElement("tipo","tipo");

          if((tipoMensaje.toString()).equals("PROFILE_REQUEST")){

            System.out.println("Se ha recibido Petición de Perfil");

            venta=new Vender(bipipe,producto,modeloVendidos);
            venta.enviarRespuestaPerfil();

            System.out.println("Se ha enviado Respuesta de Perfil");

        }
         else if ((tipoMensaje.toString()).equals("ORDER_REQUEST")){
            if (venta!=null){

                System.out.println("Se ha recibido Petición de Compra");
                MessageElement msgElement = msg.getMessageElement(SenderMessage,
SenderMessage);

fichero.EscribeFichero(rutaDirectorioPeticionesDeCompraRecibidas+nombreFichero,m
sgElement.toString());
                venta.enviarRespuestaCompra();

                modeloAnunciados.borraProducto(producto);
                System.out.println("Se ha enviado Respuesta de Compra");
                venta.ventaRealizada(nombreFichero);
                //Se borra el anuncio del producto comprado
                discovery.flushAdvertisement(mdadv);
                hilo.interrupt();


            }
            else{
               System.out.println("No se ha recibido trasacción de perfil");
            }
        }
        else{
           System.out.println("El tipo de mensaje recibido no es válido");
        }
      } catch (Exception e) {
      if (LOG.isEnabledFor(Level.DEBUG)) {
        LOG.debug(e);
      }
      return;
    }
```

```
    }

//   Método que publica un servicio en el grupo de peers de la red JXTA a la
//   que estamos unidos. Para el nombre del servicio se toma el nombre del
//   producto que se quiere anunciar
   public void PublicaServicio(ProductoXML producto){
       System.out.println("Start the Server daemon");



      try {

         ModuleClassAdvertisement mcadv = (ModuleClassAdvertisement)
                                  AdvertisementFactory.newAdvertisement(
                                    ModuleClassAdvertisement.getAdvertisementType());

         System.out.println("Nombre del producto:
"+producto.nombreProducto.getText());

         mcadv.setName(producto.nombreProducto.getText());
         mcadv.setDescription(producto.descripcion.toString());
         ModuleClassID mcID = IDFactory.newModuleClassID();
         mcadv.setModuleClassID(mcID);



          mdadv = (ModuleSpecAdvertisement)
                                 AdvertisementFactory.newAdvertisement(
                                   ModuleSpecAdvertisement.getAdvertisementType());


         mdadv.setName(producto.nombreProducto.getText());
         mdadv.setVersion("Version 1.0");
         mdadv.setCreator("sun.com");
         mdadv.setModuleSpecID(IDFactory.newModuleSpecID(mcID));
         mdadv.setSpecURI("http://www.jxta.org/Ex1");


         try {

            pipeadv = (PipeAdvertisement)
                    AdvertisementFactory.newAdvertisement(
                       PipeAdvertisement.getAdvertisementType());

            //Se establece el nombre de la tubería del servicio
            pipeadv.setName( "default");
            //Se establece el tipo de tubería del servicio
            pipeadv.setType( "JxtaUnicast" );
            //Se crea un nuevo identificador para el anuncio de la tubería.
            pipeadv.setPipeID( (ID) IDFactory.newPipeID(group.getPeerGroupID()) );
```

```
            System.out.println( "PipeID:"+ pipeadv.getPipeID());


          } catch (Exception e) {
            System.out.println("failed to read/parse pipe advertisement");
            e.printStackTrace();
            System.exit(-1);
          }


          /*Se adjunta el anuncio de Tubería al de Especificación de Módulo*/

          mdadv.setPipeAdvertisement(pipeadv);



          discovery.remotePublish(mdadv, DiscoveryService.ADV);


        } catch (Exception ex) {
          ex.printStackTrace();
          System.out.println("Server: Error publishing the module");
        }


    filaAnunciados=modeloAnunciados.insertaProducto(producto);
    }

//    Método que se ejecuta al inicializar el hilo que se crea por cada uno de
//    los servicios anunciados. Básicamente se encarga de estar a la escucha
//    y aceptar conexiones procedentes de otros peer que buscan el servicio
//    que se ha anunciado.Tras aceptar la conexion se le envía a través de esa
//    tubería bidireccional el fichero XML del producto solicitado.
    public void run(){
        System.out.println("Waiting for JxtaBidiPipe connections on JxtaServerPipe");
        try {

          serverPipe = new JxtaServerPipe(group, pipeadv);
           // we want to block until a connection is established
          serverPipe.setPipeTimeout(0);
        } catch (Exception e) {
           System.out.println("failed to bind to the JxtaServerPipe due to the following
exception");
           e.printStackTrace();
           System.exit(-1);
        }
        while (!hilo.interrupted()) {
           try {

              bipipe = serverPipe.accept();
```

```
            bipipe.setListener(this);



            if (bipipe != null ) {
              EnviaFicheroProducto(bipipe, nombreFichero);
            }
          } catch (SocketException e) {
            System.out.println ("Se finaliza el hilo");
            return;
          } catch (Exception e) {
            e.printStackTrace();
            return;
          }

      }

    }

//   Método que envía el fichero XML del producto asociado al servicio que este
//   objeto anuncia cada vez que un usuario encuentra dicho servicio anunciado.
//   El fichero se envía a través de la tubería bidireccional que se crea entre
//   este peer y el que busca el servicio.
    public void EnviaFicheroProducto(JxtaBiDiPipe bipipe, String nombreFichero){



      String texto=Fichero.LeeFichero("./Anuncios/"+nombreFichero);
      System.out.println("Se envia fichero de Producto:"+nombreFichero);

      try {

        Message msg = new Message();

        msg.addMessageElement("tipo",
                      new StringMessageElement("tipo",
                                    "PRODUCTO",
                                    null));
        msg.addMessageElement(SenderMessage,
                       new StringMessageElement(SenderMessage,
                                    texto,
                                    null));

        bipipe.sendMessage(msg);

      }
       catch (Exception ie) {
        ie.printStackTrace();
      }
```

```
    }

//   Método que inicializa el hilo cada vez que se anuncia un servicio
    public void EsperaCliente(){
       hilo.start();
    }

//   Enlaza el anuncio del servicio a la lista enlazada del modelo de la
//   tabla de productos anunciados
    public void EnlazaAnuncio(LinkedList listaAnuncios){
       listaAnuncios.add(mdadv);
    }
}
```

1.3.3. Clase TuberiaCliente

```
//
//   Fichero: TuberiaCliente.java
//   Autor: Trinitario Meseguer Calero
//   Fecha: 30/03/05
//   Descripción: Las principales funciones de esta clase son:
//      1. Establecer una conexión bidireccional con el servidor del
//      anuncio que estamos buscando.
//      2. Recibir el fichero XML del producto que estamos
//      buscando a través de esa conexión.
//      3. Almacenar dichos fichero en el directorio "Encontrados",
//      determinando previamente el nombre que le corresponde al fichero.
//      4. Recibir los mensajes cXML, ProfileResponse y OrderResponse,
//      encapsulados en mensajes JXTA que se transmiten a través de
//      la tubería bidireccional.
//      5. Transmitir un mensaje de petición de compra tras recibir
//      la respuesta de perfil y comprobar que todo es correcto.
//      Será la clase Comprar la encarga de rellenar y enviar dicho
//      mensaje OrderRequest.
//


package PFC.codigo.jxta;

import PFC.codigo.acciones.*;
import PFC.codigo.tablas.*;
import PFC.codigo.utiles.*;
import PFC.codigo.xml.*;

import net.jxta.pipe.*;
import net.jxta.discovery.*;
import net.jxta.protocol.*;
import net.jxta.document.*;
import net.jxta.endpoint.*;
import net.jxta.platform.*;
```

```
import net.jxta.util.*;
import net.jxta.id.*;
import net.jxta.peergroup.*;
import java.util.*;
import java.io.*;
import org.apache.log4j.*;



public class TuberiaCliente implements PipeMsgListener, Runnable{

   private final static Logger LOG =
Logger.getLogger(ClienteServiciosJXTA.class.getName());
   private final static String SenderMessage = "ComercioP2P";
   ModeloTablaEncontrados modeloEncontrados;
   int numFichero;
   String directorioEncontrados="./Encontrados/";
   Fichero fichero=new Fichero();
   ProductoXML producto;
   public Comprar compra;
   public int filaSeleccionada;
   public JxtaBiDiPipe bipipe;
   LinkedList listaTuberias;
    private Thread hilo;
    PeerGroup group;
    PipeAdvertisement pipeadv;

   /** Creates a new instance of TuberiaCliente */
   public TuberiaCliente(ModeloTablaEncontrados modeloEncontrados, LinkedList
listaTuberias,PeerGroup group,PipeAdvertisement pipeadv) {
       this.modeloEncontrados=modeloEncontrados;
       this.listaTuberias=listaTuberias;
       hilo=new Thread();
       this.group=group;
       this.pipeadv=pipeadv;
   }

//    Método llamado cada vez que se recibe un mensaje por la tubería
//    bidireccional por parte del servidor. Los mensaje recibidos pueden ser
//    el fichero XML del producto, una respuesta de perfil o una respuesta
//    de compra.
   public void pipeMsgEvent(net.jxta.pipe.PipeMsgEvent event) {
       MessageElement tipoMensaje;
        Message msg = null;
        String rutaFichero;
       try {
           // grab the message from the event
           msg = event.getMessage();
           if (msg == null) {
               if (LOG.isEnabledFor(Level.DEBUG)) {
```

```
            LOG.debug("Received an empty message, returning");
          }
          return;
        }
        if (LOG.isEnabledFor(Level.DEBUG)) {
          LOG.debug("Received a response");
        }
        // get the message element named SenderMessage
         tipoMensaje = msg.getMessageElement("tipo","tipo");

        System.out.println("Tipo del mensaje recibido:"+tipoMensaje.toString());
        if((tipoMensaje.toString()).equals("PRODUCTO")){

           System.out.println("Se ha recibido fichero de Producto");
           MessageElement msgElement = msg.getMessageElement(SenderMessage,
SenderMessage);
          // Get message
          if (msgElement.toString() == null) {
            System.out.println("null msg received");
          } else {
            Date date = new Date(System.currentTimeMillis());
            System.out.println("Message  :"+ msgElement.toString());


            numFichero=determinaNombreFichero(new File(directorioEncontrados));
            rutaFichero="./Encontrados/"+numFichero+".xml";
            fichero.EscribeFichero(rutaFichero,msgElement.toString());
            System.out.println("Fichero escrito");
            numFichero++;

            producto=new ProductoXML(new File(rutaFichero));


            modeloEncontrados.insertaProducto(producto);
            listaTuberias.add(this);

            for (int i=0;i<modeloEncontrados.listaProductos.size();i++){

              ProductoXML
aux=(ProductoXML)(modeloEncontrados.listaProductos.get(i));

              System.out.println(aux.nombreProducto.getText());
            }
          }
         }
        else if ((tipoMensaje.toString()).equals("PROFILE_RESPONSE")){
          System.out.println("Se ha recibido Respuesta de Perfil");
          compra.enviarPeticionCompra();
          System.out.println("Se ha enviado Peticion de Compra");
```

```
        }
        else if ((tipoMensaje.toString()).equals("ORDER_RESPONSE")){
            System.out.println("Se ha recibido Respuesta de Compra");
            modeloEncontrados.borraProducto(filaSeleccionada);
            compra.compraRealizada();


        }
        else{
            System.out.println("El tipo de mensaje recibido no es válido");
        }
      } catch (Exception e) {
        if (LOG.isEnabledFor(Level.DEBUG)) {
            LOG.debug(e);
        }
        return;
      }
   }

//   Método que determina el nombre del fichero que se recibe, que será un
//   código numérico. Para ello se examina los ficheros existentes en el
//   directorio que se le pasa como argumento y determina cuál es el código
//   más alto, de forma que incrementándolo en 1 será el nombre asignado
//   al nuevo fichero.
   int determinaNombreFichero(File directorio){

      Integer max=new Integer(0);
      Integer min;
      int comparacion;

       if(directorio.isDirectory()){

          File[] listaFicheros=directorio.listFiles();

          for(int i=0;i<listaFicheros.length;i++){

            String nombreFichero=listaFicheros[i].getName();
            String codigoFichero=nombreFichero.substring(0,nombreFichero.length()-4);
            min=new Integer(codigoFichero);
            comparacion=min.compareTo(max);
            if(comparacion>0){
               max=min;
            }
          }
      }
      else {
         System.out.println("La ruta especificada no es un directorio");
      }
```

```
      return max.intValue()+1;
    }
    void setBipipe(JxtaBiDiPipe bipipe){
       this.bipipe=bipipe;

    }

//    Método que se ejecuta al inicializar el hilo que se crea por cada uno de
//    los anuncios encontrados. Básicamente se encarga de intentar conectar
//    con el peer que anunció el servicio y estar a la escucha de los mensajes
//    procedentes de este peer a través de la tubería bidireccional que se
//    crea entre ambos.
    public void run() {

       try{
       bipipe.connect(group,
                      null,
                      pipeadv,
                      180000,
                      // register as a message listener
                      this);

          try {

            if (bipipe != null ) {
               System.out.println("JxtaBidiPipe accepted");

            }
          } catch (Exception e) {
            e.printStackTrace();

          }
       }
       catch (Exception e){
          System.out.println ("Error al conectar con el servidor");
       }

    }

}
```

1.3.4. Clase ClienteServiciosJXTA

```
//
//    Fichero: ClienteServiciosJXTA.java
//    Autor: Trinitario Meseguer Calero
//    Fecha: 27/01/05
//    Descripción: Con esta clase se implementa la parte cliente que
//       podemos encontrar en todo “peer”. Se instancia un objeto
//       de esta clase cada vez que se pulsa el botón “Buscar”,
//       del panel con el mismo nombre. Esta clase hace uso de los métodos
```

```
//      getRemoteAdvertisement y getLocalAdvertisement de la clase
//      DiscoveryService de JXTA, que trata de buscar servicios
//      cuyo nombre coincida, al menos en parte, con la cadena
//      introducida por el usuario en el panel "Buscar".Por cada anuncio
//      que encuentra crea un objeto de la clase TuberiaCliente que
//      será el encargado de establecer una conexión bidireccional
//      con el servidor. De esta forma se crearán tantas conexiones
//      como anuncios se encuentren.
//

package PFC.codigo.jxta;

import PFC.codigo.acciones.*;
import PFC.codigo.tablas.*;
import PFC.codigo.utiles.*;
import PFC.codigo.xml.*;

import net.jxta.pipe.*;
import net.jxta.discovery.*;
import net.jxta.protocol.*;
import net.jxta.document.*;
import net.jxta.endpoint.*;
import net.jxta.platform.*;
import net.jxta.util.*;
import net.jxta.id.*;
import net.jxta.peergroup.*;
import java.util.*;
import java.io.*;
import org.apache.log4j.*;
import net.jxta.peer.*;

public class ClienteServiciosJXTA {

   private PipeAdvertisement pipeadv;
   private JxtaServerPipe serverPipe;


   public PeerGroup group;
   public DiscoveryService discovery;


   Fichero fichero=new Fichero();
   int numFichero;
   ModeloTablaEncontrados modeloEncontrados;
   ProductoXML producto;
   int filaSeleccionada;
   public LinkedList listaTuberias=new LinkedList();

   /** Creates a new instance of ClienteServiciosJXTA */
   public ClienteServiciosJXTA(PlataformaJXTA plataforma,
```

```
ModeloTablaEncontrados modeloEncontrados) {
      group=plataforma.group;
      discovery=group.getDiscoveryService();
      numFichero=0;
      this.modeloEncontrados=modeloEncontrados;
   }

//   Método que busca en la red JXTA un servicio cuyo nombre coincida con la
//   cadena que se le pasa como argumento
   public void BuscarServicio(String palabraBusqueda){

      int intentos=3;
      boolean encontrado=false;
      System.out.println("Start the Client");

      System.out.println("searching Service advertisement"+palabraBusqueda);
      Enumeration enum = null;
      long parada=System.currentTimeMillis()+30000;

      try{
         PeerID peerID=group.getPeerID();
         String id=peerID.toString();
         discovery.flushAdvertisements(id,DiscoveryService.ADV);
      }
      catch(Exception e){
            System.out.println("Error al borrar los anuncios");
      }

      for (int i=0;(i<intentos)&&(encontrado==false);i++){

   //          Se busca el servicio en primer lugar de forma remota
         discovery.getRemoteAdvertisements(null
                                    , DiscoveryService.ADV
                                    , "Name"
                                    , palabraBusqueda,1, null);


         try {

   //           A continuación se busca en la caché local
               enum = discovery.getLocalAdvertisements(DiscoveryService.ADV
                                    , "Name"
                                    , palabraBusqueda);


              while ((enum != null) && enum.hasMoreElements()) {
                 enum=ConectarConServidor(enum);
                 encontrado=true;
               }
```

```
                enum=null;


                try {
//                    Esperamos 5 segundos para el nuevo intento de búsqueda
                   Thread.sleep(5000);
                } catch (Exception e) {}

             }
             catch (IOException e) {
                System.out.println ("Se ha producido una excepcion al buscar:"+e);
             }

             System.out.print(".");
       }

    }

//    Método empleado para crear una tubería bidireccional que conecte con el
//    peer que anunció el servicio que se está buscando. Por cada uno de estos
//    servicios encontrados se instancia un objeto de la clase
//    TuberiaCliente
    public Enumeration ConectarConServidor(Enumeration enum){

        System.out.println("we found the service advertisement");

//        Se obtiene el anuncio del servicio
       ModuleSpecAdvertisement mdsadv = (ModuleSpecAdvertisement)
enum.nextElement();

       try {

          StructuredTextDocument doc = (StructuredTextDocument)
                                mdsadv.getDocument(
MimeMediaType.TEXT_DEFAULTENCODING );

          StringWriter out = new StringWriter();
          doc.sendToWriter(out);
          System.out.println(out.toString());
          out.close();

//          Se obtiene el anuncio del pipe
          pipeadv = mdsadv.getPipeAdvertisement();


       } catch (Exception ex) {
          ex.printStackTrace();
          System.out.println("Client: Error sending message to the service");
       }
         try {
```

```
            System.out.println("creating the BiDi pipe");
            JxtaBiDiPipe bipipe = new JxtaBiDiPipe();
            bipipe.setReliable(true);
            System.out.println("Attempting to establish a connection");
            TuberiaCliente tuberia=new
TuberiaCliente(modeloEncontrados,listaTuberias,group,pipeadv);
             tuberia.setBipipe(bipipe);
             Thread hiloTuberia=new Thread (tuberia);
             hiloTuberia.start();


        } catch (Exception e) {
            System.out.println("failed to bind the JxtaBiDiPipe due to the following
exception");
            e.printStackTrace();
            System.exit(-1);
        }
        return enum;
    }

}
```

1.4. PAQUETE PFC.CODIGO.TABLAS

1.4.1. Clase ModeloTabla

```
//
//    Fichero: ModeloTabla.java
//    Autor: Trinitario Meseguer Calero
//    Fecha: 08/01/05
//    Descripción: Esta clase implementa la clase abstracta TableModel y
//           va a ser superclase de otras cuatro: ModeloTablaAnunciados,
//           ModeloTablaEncontrados, ModeloTablaVendidos, ModeloTablaComprados.
//           Esta superclase va a implementar los métodos abstractos de
//           TableModel además de los métodos comunes a todas sus subclases.
//

package PFC.codigo.tablas;

import PFC.codigo.acciones.*;
import PFC.codigo.jxta.*;
import PFC.codigo.utiles.*;
import PFC.codigo.xml.*;

import javax.swing.table.*;
import javax.swing.event.*;
import java.util.LinkedList;
import java.io.*;
```

```
public abstract class ModeloTabla implements TableModel{

    public LinkedList listaProductos;
    LinkedList listeners;

    /** Creates a new instance of ModeloTabla */
    public ModeloTabla() {
    }

//    Método que añade un listener al modelo de tabla
    public void addTableModelListener(TableModelListener l) {
        listeners.add (l);
    }

//    Método que devuelve la clase del objeto columna
    public Class getColumnClass(int columnIndex) {

        return String.class;
    }



//    Método que devuelve el número de filas del modelo
    public int getRowCount() {
        // Devuelve el número de productos en el modelo, es decir, el número
        // de filas en la tabla.
        return listaProductos.size();
    }

       public boolean isCellEditable(int rowIndex, int columnIndex) {

        return false;
    }


    public void removeTableModelListener(TableModelListener l) {

        // Elimina los suscriptores.
        listeners.remove(l);
    }

//    Método que asigna un determinado valor a una determinada posición de
//    la tabla, especificada por el número de fila y de columna
 public void setValueAt(Object aValue, int rowIndex, int columnIndex)
 {
     return;
 }
```

```
//   Método que borra el producto situado en una determinada fila de la
//   tabla
   public void borraProducto (int fila)
   {
      // Se borra la fila
       listaProductos.remove(fila);

      // Y se avisa a los suscriptores, creando un TableModelEvent...
      TableModelEvent evento = new TableModelEvent (this, fila, fila,
         TableModelEvent.ALL_COLUMNS, TableModelEvent.DELETE);

      // ... y pasándoselo a los suscriptores
      avisaSuscriptores (evento);
   }

//   Método que borra un determinado producto de la tabla
   public void borraProducto (ProductoXML producto)
   {
      // Se borra la fila
      int fila=-1;
      String codigoProducto=null;
      for (int i=0;i<listaProductos.size();i++){

                  ProductoXML aux=(ProductoXML)(listaProductos.get(i));
                  codigoProducto=aux.codigoProd.getText();
                  if (codigoProducto.equals(producto.codigoProd.getText())){
                     fila=i;
                  }
               }


      if (fila!=-1){
         listaProductos.remove(fila);
      }
      else{
         System.out.println("Error al borrar el producto de la tabla");
      }
      // Y se avisa a los suscriptores, creando un TableModelEvent...
      TableModelEvent evento = new TableModelEvent (this, fila, fila,
         TableModelEvent.ALL_COLUMNS, TableModelEvent.DELETE);

      // ... y pasándoselo a los suscriptores
      avisaSuscriptores (evento);
   }

//   Método que inserta un cierto producto en la tabla
    public int insertaProducto (ProductoXML nuevoProducto)
   {
      // Añade la persona al modelo
      listaProductos.add (nuevoProducto);
```

```
        // Avisa a los suscriptores creando un TableModelEvent...
        TableModelEvent evento;
        evento = new TableModelEvent (this, this.getRowCount()-1,
           this.getRowCount()-1, TableModelEvent.ALL_COLUMNS,
           TableModelEvent.INSERT);

        // ... y avisando a los suscriptores
        avisaSuscriptores (evento);

        return this.getRowCount()-1;
    }

//     Método que manda un evento a todos los suscriptores de la tabla
    private void avisaSuscriptores (TableModelEvent evento)
    {
        int i;

        // Bucle para todos los suscriptores en la lista, se llama al metodo
        // tableChanged() de los mismos, pasándole el evento.
        for (i=0; i<listeners.size(); i++)
           ((TableModelListener)listeners.get(i)).tableChanged(evento);
    }

// Método que borra todos los productos de una tabla
    public void BorraTabla(){

         listaProductos.clear();

         TableModelEvent evento;
        evento = new TableModelEvent (this, 0,
           0, TableModelEvent.ALL_COLUMNS,
           TableModelEvent.DELETE);
         // ... y avisando a los suscriptores
        avisaSuscriptores (evento);

     }

//    Método llamado cada vez que se ejecuta la aplicación y que añade a la
//    tabla todos los productos de un determinado directorio
    public void MuestraProductosAntiguos(String rutaDirectorio){

        ProductoXML producto;
        File directorio=new File(rutaDirectorio);

        if(directorio.isDirectory()){

           File[] listaFicheros=directorio.listFiles();

           for(int i=0;i<listaFicheros.length;i++){
```

```
            try{

               producto=new ProductoXML(listaFicheros[i]);
               this.insertaProducto(producto);
            }
            catch (Exception e){
               System.out.println("Error al leer el producto");
            }

         }

   }
   }

}
```

1.4.2. Clase ModeloTablaAnunciados

```
//
//   Fichero: ModeloTablaAnunciados.java
//   Autor: Trinitario Meseguer Calero
//   Fecha: 07/01/05
//   Descripción: Clase que se usa como modelo para la tabla de productos
//         anunciados, cada vez que se anuncia un nuevo producto,
//         se enlaza el objeto ProductoXML en una lista enlazada y se
//         presenta al usuario en la tabla. Los atributos del producto
//         extraídos del fichero XML que se presentan al usuario son:
//         Código del producto, Categoría del producto, Subcategoría
//         del producto, Nombre del producto, Precio, Fecha del anuncio
//



package PFC.codigo.tablas;

import PFC.codigo.acciones.*;
import PFC.codigo.jxta.*;
import PFC.codigo.utiles.*;
import PFC.codigo.xml.*;

import javax.swing.table.*;
import javax.swing.event.*;
import java.util.LinkedList;
import javax.swing.table.*;

public class ModeloTablaAnunciados extends ModeloTabla implements TableModel {

    /** Lista con los datos. Cada elemento de la lista es una instancia de
    * ProductoXML */
```

```
/** Lista de suscriptores. El JTable será un suscriptor de este modelo de
 * datos */


/** Creates a new instance of Class */


public ModeloTablaAnunciados() {
   listaProductos = new LinkedList();
   listeners = new LinkedList();
}


 public String getColumnName(int columnIndex)
{
   // Devuelve el nombre de cada columna. Este texto aparecerá en la
   // cabecera de la tabla.
   switch (columnIndex)
   {
      case 0:
         return "Codigo Producto";
      case 1:
         return "Categoria";
      case 2:
         return "Subcategoria";
      case 3:
         return "Producto";
      case 4:
         return "Precio";
      case 5:
         return "Fecha Anuncio";
      default:
         return null;
   }
}
 public Object getValueAt(int rowIndex, int columnIndex) {
    ProductoXML aux;

   // Se obtiene el producto de la fila indicada
   aux = (ProductoXML)(listaProductos.get(rowIndex));

   // Se obtiene el campo apropiado según el valor de columnIndex
   switch (columnIndex)
   {
      case 0:
         return aux.codigoProd.getText();
      case 1:
         return aux.categoria.getText();
      case 2:
         return aux.subcategoria.getText();
```

```
            case 3:
               return aux.nombreProducto.getText();
            case 4:
               return aux.precio.getText();
            case 5:
               return aux.fechaAnuncio.getText();
            default:
               return null;
         }
      }
       public int getColumnCount() {
         // Devuelve el número de columnas del modelo, que coincide con el
        // número de datos que tenemos de cada persona.
        return 6;
     }


}
```

1.4.3. Clase ModeloTablaEncontrados

```
//
//   Fichero: ModeloTablaEncontrados.java
//   Autor: Trinitario Meseguer Calero
//   Fecha: 08/01/05
//   Descripción: Esta clase servirá de modelo para la tabla de productos
//      encontrados. La lista enlazada de productos se borra en cada
//      búsqueda, de forma que solo aparezcan en la tabla los productos
//      correspondientes a la última búsqueda. Los atributos que nos
//      interesaba presentar de los productos encontrados son: Categoría
//      del producto, Subcategoría del producto, Nombre del producto,
//      Proveedor, Precio, Fecha del anuncio.
//

package PFC.codigo.tablas;

import PFC.codigo.acciones.*;
import PFC.codigo.jxta.*;
import PFC.codigo.utiles.*;
import PFC.codigo.xml.*;

import javax.swing.table.*;
import javax.swing.event.*;
import java.util.LinkedList;
import javax.swing.table.*;

public class ModeloTablaEncontrados extends ModeloTabla implements TableModel {

   /** Creates a new instance of ModeloTablaEncontrados */
   public ModeloTablaEncontrados() {
      listaProductos = new LinkedList();
```

```
    listeners = new LinkedList();
}
 public String getColumnName(int columnIndex)
{
    // Devuelve el nombre de cada columna. Este texto aparecerá en la
    // cabecera de la tabla.
    switch (columnIndex)
    {
        case 0:
            return "Categoria";
        case 1:
            return "Subcategoria";
        case 2:
            return "Producto";
        case 3:
            return "Proveedor";
        case 4:
            return "Precio";
        case 5:
            return "Fecha Anuncio";
        default:
            return null;
    }
}
 public Object getValueAt(int rowIndex, int columnIndex) {
     ProductoXML aux;

    // Se obtiene el producto de la fila indicada
    aux = (ProductoXML)(listaProductos.get(rowIndex));

    // Se obtiene el campo apropiado según el valor de columnIndex
    switch (columnIndex)
    {
        case 0:
            return aux.categoria.getText();
        case 1:
            return aux.subcategoria.getText();
        case 2:
            return aux.nombreProducto.getText();
        case 3:
            return aux.empresa.getText();
        case 4:
            return aux.precio.getText();
        case 5:
            return aux.fechaAnuncio.getText();
        default:
            return null;
    }
 }
   public int getColumnCount() {
```

```
        // Devuelve el número de columnas del modelo, que coincide con el
        // número de datos que tenemos de cada persona.
        return 6;
    }

}
```

1.4.4. Clase ModeloTablaVendidos

```
//
//   Fichero: ModeloTablaVendidos.java
//   Autor: Trinitario Meseguer Calero
//   Fecha: 07/03/05
//   Descripción: Modelo de la tabla de productos vendidos. Una vez que se
//       realiza la venta de un determinado producto, se hará uso de esta
//       clase para añadir el producto vendido a la lista enlazada
//       correspondiente y para que sea mostrado en la tabla de
//       productos vendidos. Los atributos del producto que se
//       muestran en esta tabla son: Código del producto, Categoría del
//       producto, Subcategoría del producto, Nombre del producto.
//

package PFC.codigo.tablas;

import PFC.codigo.acciones.*;
import PFC.codigo.jxta.*;
import PFC.codigo.utiles.*;
import PFC.codigo.xml.*;

import java.util.LinkedList;
import javax.swing.table.*;

public class ModeloTablaVendidos  extends ModeloTabla implements TableModel {

    /** Creates a new instance of ModeloTablaVendidos */
    public ModeloTablaVendidos() {
        listaProductos = new LinkedList();
        listeners = new LinkedList();
    }

        // Devuelve el nombre de cada columna. Este texto aparecerá en la
        // cabecera de la tabla.
    public String getColumnName(int columnIndex) {
         switch (columnIndex)
        {
            case 0:
                return "Codigo Producto";
            case 1:
                return "Categoria";
            case 2:
                return "Subcategoria";
```

```
            case 3:
               return "Producto";
            case 4:
               return "Precio";

            default:
               return null;
      }
      }

      public Object getValueAt(int rowIndex, int columnIndex) {
          ProductoXML aux;

         // Se obtiene el producto de la fila indicada
         aux = (ProductoXML)(listaProductos.get(rowIndex));

         // Se obtiene el campo apropiado según el valor de columnIndex
         switch (columnIndex)
         {
            case 0:
               return aux.codigoProd.getText();
            case 1:
               return aux.categoria.getText();
            case 2:
               return aux.subcategoria.getText();
            case 3:
               return aux.nombreProducto.getText();
            case 4:
               return aux.precio.getText();

            default:
               return null;
         }

      }
        public int getColumnCount() {
          // Devuelve el número de columnas del modelo, que coincide con el
         // número de datos que tenemos de cada persona.
         return 5;
      }

      }
```

1.4.5. Clase ModeloTablaComprados

```
//
//    Fichero: ModeloTablaComprados.java
//    Autor: Trinitario Meseguer Calero
//    Fecha: 07/03/05
//    Descripción: Clase que servirá de modelo para la tabla de productos
//       comprados. En este caso los atributos del producto que se extraen
```

```
//      del fichero XML para presentarlos al usuario son:Categoría
//      del producto, Subcategoría del producto, Nombre del producto,
//      Proveedor, Precio del producto
//


package PFC.codigo.tablas;

import PFC.codigo.acciones.*;
import PFC.codigo.jxta.*;
import PFC.codigo.utiles.*;
import PFC.codigo.xml.*;

import java.util.LinkedList;
import javax.swing.table.*;

public class ModeloTablaComprados extends ModeloTabla implements TableModel{

   /** Creates a new instance of ModeloTablaComprados */
   public ModeloTablaComprados() {
      listaProductos = new LinkedList();
      listeners = new LinkedList();
   }

   // Devuelve el nombre de cada columna. Este texto aparecerá en la
   // cabecera de la tabla.
   public String getColumnName(int columnIndex) {
      switch (columnIndex)
      {
         case 0:
            return "Categoria";
         case 1:
            return "Subcategoria";
         case 2:
            return "Producto";
         case 3:
            return "Proveedor";
         case 4:
            return "Precio";

         default:
            return null;
      }
   }

  public Object getValueAt(int rowIndex, int columnIndex) {
       ProductoXML aux;

      // Se obtiene la producto de la fila indicada
      aux = (ProductoXML)(listaProductos.get(rowIndex));
```

```
      // Se obtiene el campo apropiado según el valor de columnIndex
      switch (columnIndex)
      {
         case 0:
            return aux.categoria.getText();
         case 1:
            return aux.subcategoria.getText();
         case 2:
            return aux.nombreProducto.getText();
         case 3:
            return aux.empresa.getText();
         case 4:
            return aux.precio.getText();

         default:
            return null;
      }
   }
    public int getColumnCount() {
       // Devuelve el número de columnas del modelo, que coincide con el
      // número de datos que tenemos de cada persona.
      return 5;
   }

}
```

1.4.6. Clase MuestraProducto

```
//
//   Fichero: MuestraProducto.java
//   Autor: Trinitario Meseguer Calero
//   Fecha: 11/01/05
//   Descripción: Se trata de una subclase de JFrame del paquete SWING,
//      es la clase responsable de que al seleccionar cualquier fila de
//      cualquiera de las tablas de nuestra aplicación se despliegue un
//      panel donde podamos ver al completo el fichero XML del producto
//      seleccionado. Consta de un objeto JTextArea que se incluye en un
//      objeto JScrollPane y sobre el que se vuelca la información del
//      objeto ProductoXML en forma de String.
//

package PFC.codigo.tablas;

import PFC.codigo.acciones.*;
import PFC.codigo.jxta.*;
import PFC.codigo.utiles.*;
import PFC.codigo.xml.*;

import java.awt.event.*;
```

```
public class MuestraProducto extends javax.swing.JFrame {

    /** Creates new form MuestraProducto */
    public MuestraProducto(ProductoXML producto) {
        initComponents();
        String texto=producto.toString();
        textoAnuncio.append(texto);
    }

//    Método que inicializa los componentes del panel que muestra en detalle
//    el fichero XML asociado al producto seleccionado
    private void initComponents() {
        panelDatosProducto = new javax.swing.JPanel();
        panelDeslizanteDatosProducto = new javax.swing.JScrollPane();
        textoAnuncio = new javax.swing.JTextArea();
        etDatosProducto = new javax.swing.JLabel();
        botonSalir = new javax.swing.JButton();

        setTitle("Datos del Producto");
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                exitForm(evt);
            }
        });

        panelDatosProducto.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        panelDeslizanteDatosProducto.setViewportView(textoAnuncio);

        panelDatosProducto.add(panelDeslizanteDatosProducto, new
org.netbeans.lib.awtextra.AbsoluteConstraints(30, 40, 470, 240));

        etDatosProducto.setText("DATOS DEL PRODUCTO");
        panelDatosProducto.add(etDatosProducto, new
org.netbeans.lib.awtextra.AbsoluteConstraints(200, 20, -1, -1));

        botonSalir.setText("Salir");
        botonSalir.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                botonSalirActionPerformed(evt);
            }
        });

        panelDatosProducto.add(botonSalir, new
org.netbeans.lib.awtextra.AbsoluteConstraints(440, 310, -1, -1));

        getContentPane().add(panelDatosProducto, java.awt.BorderLayout.CENTER);

        pack();
    }
```

```
//    Método llamado al pulsar el botón "Salir" del panel que muestra los
//    datos del producto, lanza un evento que cierra el panel
    private void botonSalirActionPerformed(java.awt.event.ActionEvent evt) {

        WindowEvent eventoCierreVentana=new WindowEvent(this,
WindowEvent.WINDOW_CLOSING);
        this.dispatchEvent(eventoCierreVentana);
    }

//    Método que devuelve el control del programa al objeto que lo llamó
    private void exitForm(java.awt.event.WindowEvent evt) {
        return;
    }


    // Variables declaration - do not modify
    private javax.swing.JButton botonSalir;
    private javax.swing.JLabel etDatosProducto;
    public javax.swing.JPanel panelDatosProducto;
    private javax.swing.JScrollPane panelDeslizanteDatosProducto;
    private javax.swing.JTextArea textoAnuncio;
    // End of variables declaration

}
```

1.4.7. Clase MuestraProductoEncontrado

```
//
//    Fichero: MuestraProductosEncontrados.java
//    Autor: Trinitario Meseguer Calero
//    Fecha: 16/01/05
//    Descripción: la clase MuestraProducto se instancia cada vez que
//        seleccionamos una fila de cualquier tabla de nuestra aplicación,
//        a excepción de la tabla de productos encontrados. Para esta tabla
//        se instancia un objeto de la clase MuestraProductoEncontrado que es
//        subclase de la anterior.La funcionalidad es la misma, mostrar al
//        completo el fichero XML asociado al producto que seleccionamos.
//        Pero si observamos aparece un botón más que en el panel de
//        MuestraProducto, se trata del botón “Comprar”. Al pulsar este botón
//        se instancia un objeto de la clase DatosComprador y se muestra dicho
//        panel a la vez que se cierra el panel que muestra el producto.
//

package PFC.codigo.tablas;

import PFC.codigo.acciones.*;
import PFC.codigo.jxta.*;
import PFC.codigo.utiles.*;
import PFC.codigo.xml.*;
```

```
import javax.swing.*;
import net.jxta.util.*;
import java.awt.event.*;

public class MuestraProductoEncontrado extends MuestraProducto {
   ProductoXML producto;
   ModeloTablaComprados modeloComprados;
   TuberiaCliente tuberia;

//   Constructor de la clase MuestraProductoEncontrado. Es una subclase
//   de muestra producto lo único que añade es el botón "Comprar"
   public MuestraProductoEncontrado(ProductoXML producto,TuberiaCliente tuberia,
ModeloTablaComprados modeloComprados) {


       super(producto);
       this.tuberia=tuberia;
       this.producto=producto;

       this.modeloComprados=modeloComprados;
       JButton botonComprar = new JButton();
       botonComprar.setText("COMPRAR");
       botonComprar.addActionListener(new java.awt.event.ActionListener() {
          public void actionPerformed(java.awt.event.ActionEvent evt) {
             botonCompararActionPerformed(evt);
          }
       });
       panelDatosProducto.add(botonComprar, new
org.netbeans.lib.awtextra.AbsoluteConstraints(190, 290, 130, 40));


   }

//   Método llamado al pulsar el botón comprar del panel que se despliega al
//   seleccionar un determinado producto de la tabla de productos encontrados.
//   Se instancia un objeto de la clase DatosComprador y se muestra el panel
//   correspondiente, a la vez que se cierra el panel que muestra el detalle
//   del producto.
    private void botonCompararActionPerformed(java.awt.event.ActionEvent evt) {

       DatosComprador panel=new DatosComprador(producto,tuberia,
modeloComprados);
       panel.show();

        WindowEvent eventoCierreVentana=new WindowEvent(this,
WindowEvent.WINDOW_CLOSING);
        this.dispatchEvent(eventoCierreVentana);

   }
```

```
//    Método que devuelve el control del programa al objeto que lo llamó
      private void exitForm(java.awt.event.WindowEvent evt) {
      return;
   }
}
```

1.5. PAQUETE PFC.CODIGO.UTILES

1.5.1. Clase FactoriaAleatoria

```
//
//    Fichero: FactoriaAleatoria.java
//    Autor: Trinitario Meseguer Calero
//    Fecha: 08/01/05
//    Descripción: Contiene la clase FactoriaAleatoria, que proporciona métodos
//           de generación de números y cadenas aleatorias.
//


package PFC.codigo.utiles;

import PFC.codigo.acciones.*;
import PFC.codigo.jxta.*;
import PFC.codigo.tablas.*;
import PFC.codigo.xml.*;

import java.util.Random;
import java.util.Calendar;
import java.util.GregorianCalendar;

/**
 * Clase que contiene métodos de generación de números y cadenas
 * <code>String</code> numéricas aleatorias.
 *
 *
 */
public abstract class FactoriaAleatoria {

   /**
    * Constante de limite superior de generación de números aleatorios. Ningún
    * valor generado será mayor que este número.
    */
   protected final static int LIMITE = 10000000;

   /** Variable de clase para la generación de números aleatorios */
   private static Random random;

   /*
    * Inicializador estático de la clase. Crea un nuevo objeto
```

```
    * Random y le asigna una semilla basada en los milisegundos
    * de la hora actual. De esta manera se consigue una generación aleatoria
    * eficaz, ya que la semilla solo se crea una sola vez, la primera en que
    * se usa la clase.
    */
   static {
      random = new Random();

      Calendar calendario = new GregorianCalendar();
      random.setSeed( calendario.getTimeInMillis() );
   }

   /**
    * Genera un número aleatorio comprendido entre 0 y <code>LIMITE</code>
    *
    * @return un entero aleatorio comprendido entre 0 y <code>LIMITE</code>
    */
   public static int numeroAleatorio() {
      return random.nextInt(LIMITE);
   }

   /**
    * Genera una cadena de caracteres numéricos aleatoria comprendido entre 0
    * y <code>LIMITE</code>.
    *
    * @return una cadena <code>String</code> de caracteres numéricos aleatoria
    *         comprendidos entre 0 y <code>LIMITE</code>.
    */
   public static String cadenaNumericaAleatoria() {
      int i = random.nextInt(LIMITE);
      return String.valueOf(i);
   }

}
```

1.5.2. Clase FechaISO8601

```
//
//   Fichero: FechaISO8601.java
//   Autor: Trinitario Meseguer Calero
//   Fecha: 08/01/05
//   Descripción: Contiene la clase FechaISO8601, que proporciona métodos
//            de generación de fechas ISO8601.
//


package PFC.codigo.utiles;

import PFC.codigo.acciones.*;
import PFC.codigo.jxta.*;
import PFC.codigo.tablas.*;
```

```
import PFC.codigo.xml.*;

import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.StringTokenizer;
import java.util.TimeZone;

/**
 * Colección de métodos útiles para trabajar con fechas en formato ISO-8601.
 * <br>
 * El formato ISO-8601 especifica la representación de fecha y hora para
 * estandarizar la notación a la hora de las comunicaciones internacionales.
 * Además, su estructura es idónea para su empleo en computadoras. <br>
 * <br>El estandar ISO8601 admite una amplia variedad de formatos según
 * situaciones concretas. En esta clase se implementa un caso particular
 * respecto a la hora UTC (Universal Time). El formato consta de:
 *       YYYY-MM-DDThh:mm:ss.000+DD:DD
 *     ó YYYY-MM-DDThh:mm:ss.000-DD:DD
 * donde :
 *       YYYY  : año con 4 cifras
 *       MM    : mes con 2 cifras
 *       DD    : dia con 2 cifras
 *       T     : indica que lo siguiente es la hora
 *       hh    : hora con 2 cifras
 *       mm    : minutos con 2 cifras
 *       ss    : segundos con 2 cifras
 *       000   : milisegundos con 3 cifras
 *       DD:DD : desfase en horas respecto a la hora UTC
 *
 * <b>Nota:</b>
 * La hora UTC se considera respecto al meridiano de Greenwich. También es
 * conocida por GMT (Greenwich Mean Time).
 *
 *
 */
public abstract class FechaISO8601 {

   /**
    * Constante que define el caracter de desfase horario respecto a UTC.
    */
   protected static final String MAS_TOKEN = "+";

   /**
    * Constante que define el caracter separador entre los digitos de la fecha
    * y de desfase horario respecto a UTC.
    */
   protected static final String MENOS_TOKEN = "-";

   /**
```

```
 * Constante que define el caracter separador entre los digitos de la fecha
 * y los digitos de la hora.
 */
protected static final String T_TOKEN = "T";

/**
 * Constante que define el caracter separador entre los digitos de la hora.
 */
protected static final String DOSPUNTOS_TOKEN = ":";

/**
 * Constante que define el caracter separador entre los digitos de la hora
 * y los digitos de los milisegundos.
 */
protected static final String PUNTO_TOKEN = ".";

/** Constante que define el caracter indicador de hora UTC */
protected static final String Z_TOKEN = "Z";

/**
 * Convierte un entero en una cadena de 2 caracteres representando su valor.
 * Si el entero es menor que 10, se le añade un 0. Si es mayor que 10 pero
 * menor que 100, se deja tal cual. Si es mayor de 100, el método usará las
 * 2 primeras cifras significativas del entero.
 *
 */
protected static String convertirEnDosDigitos(int i) {
   //Se halla el valor absoluto del entero
   int j = (i > 0) ? i
                : ( i * (-1) );
   //Se obtiene la representación del valor absoluto del entero como cadena
   String aux = String.valueOf(j);
   /*
    * Si el entero es mayor que 99, se cogen las dos primeras cifras
    * significativas.
    */
   if (j > 99) {
      aux = aux.substring(0, 2);
   }
   /*
    * Si el entero es menor que 10, se le añade un cero delante. Si no,
    * se deja igual.
    */
   return ( (j >=0 && j < 10) ? ( "0" + aux ) : aux );
}

/**
 * Convierte un entero en una cadena de 3 caracteres representando su valor.
 * Si el entero es menor que 100, se le añade un 0 a la izquierda. Si es
 * mayor que 99, se deja tal cual.
```

```
     *
     */
    protected static String convertirEnTresDigitos(int i) {
        //Se halla el valor absoluto del entero
        int j = (i > 0) ? i
                        : ( i * (-1) );
        //Se obtiene la representación del valor absoluto del entero como cadena
        String aux = String.valueOf(j);
        /*
         * Si el entero es menor que 99, se le añade un cero delante. Si no,
         * se deja igual.
         */
        return (  (j < 100) ? ( "0" + aux ) : aux );
    }

    /**
     * Obtiene la representación de la fecha y hora actuales en formato
     * ISO8601.
     *
     * @return  un <code>String</code> en formato ISO8601 con el tiempo actual.
     */
    public static String fechaISO() {
        return fechaISO( new GregorianCalendar() );
    }

    /**
     * Obtiene la representación en formato ISO8601 de la fecha especificada.
     *
     */
    public static String fechaISO(Calendar fecha) {
        StringBuffer buffer = new StringBuffer();

        /* Obtenemos un objeto Calendar con la hora local */
        Calendar calendario = new GregorianCalendar();
        calendario.setTime( fecha.getTime() );

        // Obtenemos el año y lo añadimos a la cadena buffer
        buffer.append(calendario.get(Calendar.YEAR));
        buffer.append(MENOS_TOKEN);
        //Obtenemos el mes y lo añadimos al buffer
        buffer.append(convertirEnDosDigitos(calendario.get(Calendar.MONTH) + 1));
        buffer.append(MENOS_TOKEN);
        //Obtenemos el dia y lo añadimos al buffer

buffer.append(convertirEnDosDigitos(calendario.get(Calendar.DAY_OF_MONTH)));
        buffer.append(T_TOKEN);
        //Obtenemos la hora y lo añadimos al buffer

buffer.append(convertirEnDosDigitos(calendario.get(Calendar.HOUR_OF_DAY)));
        buffer.append(DOSPUNTOS_TOKEN);
```

```
        //Obtenemos los minutos y los añadimos al buffer
        buffer.append(convertirEnDosDigitos(calendario.get(Calendar.MINUTE)));
        buffer.append(DOSPUNTOS_TOKEN);
        //Obtenemos los segundos y los añadimos al buffer
        buffer.append(convertirEnDosDigitos(calendario.get(Calendar.SECOND)));
        buffer.append(PUNTO_TOKEN);
        //Obtenemos los milisegundos y los añadimos al buffer

buffer.append(convertirEnTresDigitos(calendario.get(Calendar.MILLISECOND)));

        //Se obtiene el offset respecto a la hora UTC en horas.
        int offset = calendario.get(Calendar.ZONE_OFFSET)/(60*60*1000);
        if(offset < 0) {
           buffer.append(MENOS_TOKEN);
        } else {
           buffer.append(MAS_TOKEN);
        }
        buffer.append(convertirEnDosDigitos(offset));
        buffer.append(DOSPUNTOS_TOKEN + "00");

        return buffer.toString();
     }

     /**
      * Obtiene un objeto <code>Calendar</code> a partir de una cadena
      * <code>String</code> que contiene una fecha en formato ISO8601.
      *
      */
     public static Calendar fecha(String fechaISO) {
        //Se crea un objeto Calendar apropiado para UTC
        Calendar calendario = new GregorianCalendar();
        //Se usan tokens de los caracteres separadores de ISO8601
        StringTokenizer st = new StringTokenizer(fechaISO,
             MAS_TOKEN + MENOS_TOKEN + T_TOKEN + Z_TOKEN +
DOSPUNTOS_TOKEN +
             PUNTO_TOKEN, false);

        try {
           /*
            * Se obtiene la cadena que representa al año buscando hasta
            * el siguiente token. Esto se repite para los meses, dias, etc.
            */
           int agno = Integer.parseInt(st.nextToken());
           calendario.set(Calendar.YEAR, agno);

           //El mes
           int mes = Integer.parseInt(st.nextToken()) -1;
           calendario.set(Calendar.MONTH, mes);

           //El dia
```

```
            int dia = Integer.parseInt(st.nextToken());
            calendario.set(Calendar.DAY_OF_MONTH, dia);

            //La hora
            int hora = Integer.parseInt(st.nextToken());
            calendario.set(Calendar.HOUR_OF_DAY, hora);

            //Los minutos
            int minutos = Integer.parseInt(st.nextToken());
            calendario.set(Calendar.MINUTE, minutos);

            //Los segundos
            int segundos = Integer.parseInt(st.nextToken());
            calendario.set(Calendar.SECOND, segundos);

            /* No se procesan los milisegundos */
         } catch (Exception e) {
            //No se procesa nada, sólo se devuelve null
            calendario = null;
         }

         return calendario;
      }

}
```

1.5.3. Clase UtilesDirectorio

```
//
//    Fichero: UtilesDirectorio.java
//    Autor: Trinitario Meseguer Calero
//    Fecha: 08/01/05
//    Descripción: Contiene la clase UtilesDirectorio, que proporciona un método
//          de borrado de directorios.
//

package PFC.codigo.utiles;

import PFC.codigo.acciones.*;
import PFC.codigo.jxta.*;
import PFC.codigo.tablas.*;
import PFC.codigo.xml.*;

import java.io.File;

/**
 * Clase que proporciona métodos relacionados con el manejo de directorios.
 *
 *
 */
public abstract class UtilesDirectorio {
```

```
   /*
    * Esta clase sólo implementa 1 método de borrado de directorio, debido a
    * que el proyecto no ha requerido más métodos de manejo de archivos y
    * directorios.
    */

   /**
    * Borra un directorio del sistema de archivos, usando un algoritmo
    * recursivo para eliminar los archivos y directorios contenidos.
    *
    */
   public static boolean borrarDirectorio(String ruta) {
      boolean resultado = true;
      File dir = new File(ruta);

      //Se comprueba primero que exista el directorio
      if (dir != null) {
         //Obtenemos todos los archivos y directorios del directorio actual
         File[] archivos = dir.listFiles();

         if (archivos != null) {
            for (int i = 0; i < archivos.length; i++) {
               File archivo = archivos[i];

               //Si es archivo, simplemente se borra
               if ( archivo.isFile() == true ) {
                  if (!archivo.delete()) {
                     resultado = false;
                  }
               } else {
                  //Si es directorio,llamamos a este metodo recursivamente
                  if ( borrarDirectorio(archivo.getPath()) == false ) {
                     resultado = false;
                  }
               }
            }
            //Se borra el directorio
            if ( dir.delete() == false )
               resultado = false;
         }
      }
      return resultado;
   }

}
```

1.5.4. Clase Fichero

```
//
//    Fichero: Fichero.java
//    Autor: Trinitario Meseguer Calero
```

```
//    Fecha: 18/01/05
//    Descripción: Contiene la clase Fichero, que proporciona métodos
//                 para leer y escribir en ficheros.
//

package PFC.codigo.utiles;

import PFC.codigo.acciones.*;
import PFC.codigo.jxta.*;
import PFC.codigo.tablas.*;
import PFC.codigo.xml.*;

import java.io.*;

public class Fichero {
       static String cadena;
       static int numCaracteres;
       static byte buffer[]=null;
       static int i;
       static FileInputStream fichero_in;
       static FileOutputStream fichero_out;


   /** Creates a new instance of Fichero */
   public Fichero() {
   }


//  Método que lee el fichero cuyo nombre se le pasa como argumento y lo
//  devuelve en forma de String
   public static String LeeFichero(String nombreFichero){

       i=0;
      try{
          fichero_in= new FileInputStream(nombreFichero);
          numCaracteres=fichero_in.available();
          buffer=new byte[numCaracteres];
          while (i<numCaracteres){
             buffer[i]=(byte)fichero_in.read();
             i++;


          }
          fichero_in.close();
      } catch(Exception e){

         System.out.println("Excepcion"+e);

       }
```

```
        cadena=new String(buffer);
        return cadena;

    }

//    Metodo que recibe un texto en forma de String y genera un fichero cuyo
//    nombre también se le pasa como parámetro y de contenido ese texto
    public static void EscribeFichero(String nombreFichero, String texto){
          try{
        fichero_out=new FileOutputStream(nombreFichero);
        buffer=texto.getBytes();
        fichero_out.write(buffer);
        fichero_out.close();
        } catch(Exception e){

          System.out.println("Excepcion"+e);

        }

    }

//    Método al que se le pasa la ruta de un determinado directorio y
//    borra todos los fichero de dicho directorio
    public static void BorraContenidoDirectorio(String rutaDirectorio){

        File directorio=new File(rutaDirectorio);
        if(directorio.isDirectory()){
           File[] listaFicheros=directorio.listFiles();
           for(int i=0;i<listaFicheros.length;i++){
              listaFicheros[i].delete();
           }
        }
        else {
           System.out.println("La ruta especificada no es un directorio");
        }
    }
}
```

1.6. PAQUETE PFC.CODIGO

1.6.1. Clase Comercio

```
//
//    Fichero: Comercio.java
//    Autor: Trinitario Meseguer Calero
//    Fecha: 24/11/04
//    Descripción: Fichero que contiene la clase principal de la aplicación cuya
//          función es inicializar los componentes gráficos del programa,
```

```
//          conectarse a la red JXTA y permanecer a la escucha de determinados
//          eventos que genera el usuario al pulsar los botones de los distintos
//          paneles.
//

package PFC.codigo;

import PFC.codigo.acciones.*;
import PFC.codigo.jxta.*;
import PFC.codigo.tablas.*;
import PFC.codigo.utiles.*;
import PFC.codigo.xml.*;

import org.jdom.*;
import java.io.*;
import net.jxta.util.*;
import java.util.*;


public class Comercio extends javax.swing.JFrame {


      Categorias ObjCategorias=new Categorias();
      public PlataformaJXTA plataforma=new PlataformaJXTA();
      ModeloTablaAnunciados modeloAnunciados=new ModeloTablaAnunciados();
      ModeloTablaEncontrados modeloEncontrados=new ModeloTablaEncontrados();
      ModeloTablaComprados modeloComprados=new ModeloTablaComprados();
      ModeloTablaVendidos modeloVendidos=new ModeloTablaVendidos();

      public int codigoProducto=0;
      ClienteServiciosJXTA cliente;
      String rutaProductosAnunciados="./Anuncios/";
      String rutaProductosComprados="./Comprados/";
      String rutaProductosVendidos="./Vendidos/";
      LinkedList listaAnuncios=new LinkedList();
      String nombreGrupo="ComercioElectronico1.0";

      Anunciar anunciosAntiguos=new Anunciar(null, null,
plataforma,modeloAnunciados, modeloVendidos,listaAnuncios);

//   Contructor de la clase Comercio, se encarga de conectar la aplicación a
//   la red JXTA, e inicializar los modelos de tablas que componen
//   la aplicación.
   public Comercio() {

      plataforma.limpiarCache();
      plataforma.startJXTA();
      System.out.println("Espera mientras se conecta al rendezvous");
      plataforma.conectarConRendezvous(10, 6);
```

```
        String aux = (plataforma.estaConectadoConRendezvous() ? "SI": "NO");
        System.out.println(aux + " estoy conectado al exterior de la red local.");
        System.out.println("Espera mientras se conecta a la red \"Comercio Electronico
1.0\"");
        plataforma.entrarEnGrupo(nombreGrupo, "Grupo de ComercioElectronico1.0");

        initComponents();
        anunciosAntiguos.AnunciarProductosAntiguos();
        modeloComprados.MuestraProductosAntiguos(rutaProductosComprados);
        modeloVendidos.MuestraProductosAntiguos(rutaProductosVendidos);

codigoProducto=DeterminaCodigoProducto(modeloAnunciados,modeloVendidos);
        CampoCodProducto.setText(String.valueOf(codigoProducto));


    }

//    Método que inicializa todos los componentes de la interfaz gráfica de
//    la aplicación
    private void initComponents() {
        java.awt.GridBagConstraints gridBagConstraints;

        ContenedorPaneles = new javax.swing.JTabbedPane();
        PanelBienvenida = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        PanelAnunciar = new javax.swing.JPanel();
        PanelDatosVendedor = new javax.swing.JPanel();
        CampoLocalidad = new javax.swing.JTextField();
        CampoDNI = new javax.swing.JTextField();
        EtCIF = new javax.swing.JLabel();
        CampoNombre = new javax.swing.JTextField();
        EtTelefono = new javax.swing.JLabel();
        CampoCP = new javax.swing.JTextField();
        CampoCIF = new javax.swing.JTextField();
        EtEmpresa = new javax.swing.JLabel();
        CampoDireccion = new javax.swing.JTextField();
        EtDNI = new javax.swing.JLabel();
        CampoProvincia = new javax.swing.JTextField();
        EtCP = new javax.swing.JLabel();
        CampoTelefono = new javax.swing.JTextField();
        EtDireccion = new javax.swing.JLabel();
        CampoApellidos = new javax.swing.JTextField();
        EtApellidos = new javax.swing.JLabel();
        EtPoblacion = new javax.swing.JLabel();
        CampoEmpresa = new javax.swing.JTextField();
        EtProvincia = new javax.swing.JLabel();
        EtNombre = new javax.swing.JLabel();
        PanelDatosProducto = new javax.swing.JPanel();
```

```
EtCodProducto = new javax.swing.JLabel();
EtCategoria = new javax.swing.JLabel();
EtSubcategoria = new javax.swing.JLabel();
EtProducto = new javax.swing.JLabel();
EtDescripcion = new javax.swing.JLabel();
CampoCodProducto = new javax.swing.JTextField();
CampoNombreProducto = new javax.swing.JTextField();
EtPrecio = new javax.swing.JLabel();
CampoPrecio = new javax.swing.JTextField();
EtEuro = new javax.swing.JLabel();
CampoOpcCategoria = new javax.swing.JComboBox();
CampoOpcSubcategoria = new javax.swing.JComboBox();
TextoDescripcion = new javax.swing.JTextPane();
BotonAnunciar = new javax.swing.JButton();
CampoFechaAnuncio = new javax.swing.JTextField();
EtFechaAnuncio = new javax.swing.JLabel();
PanelDatosRed = new javax.swing.JPanel();
EtNombreDominio = new javax.swing.JLabel();
EtIdentidadDominio = new javax.swing.JLabel();
CampoNombreDominio = new javax.swing.JTextField();
CampoIdentidadDominio = new javax.swing.JTextField();
jLabel4 = new javax.swing.JLabel();
PanelBuscar = new javax.swing.JPanel();
BotonBuscar = new javax.swing.JButton();
EtProdEncontrados = new javax.swing.JLabel();
jScrollPane1 = new javax.swing.JScrollPane();
TablaProdEncontrados = new javax.swing.JTable();
CamProdBusc = new javax.swing.JTextField();
EtProdBusc = new javax.swing.JLabel();
PanelCompras = new javax.swing.JPanel();
jScrollPane3 = new javax.swing.JScrollPane();
TablaProdComprados = new javax.swing.JTable();
PanelVentas = new javax.swing.JPanel();
jScrollPane4 = new javax.swing.JScrollPane();
TablaProdVendidos = new javax.swing.JTable();
PanelProdAnunciados = new javax.swing.JPanel();
jScrollPane2 = new javax.swing.JScrollPane();
TablaProdAnunciados = new javax.swing.JTable();

setTitle("Comercio Electr\u00f3nico 1.0");
setResizable(false);
addWindowListener(new java.awt.event.WindowAdapter() {
   public void windowClosing(java.awt.event.WindowEvent evt) {
      exitForm(evt);
   }
});

ContenedorPaneles.setTabPlacement(javax.swing.JTabbedPane.LEFT);
ContenedorPaneles.setMaximumSize(new java.awt.Dimension(1000, 800));
ContenedorPaneles.setMinimumSize(new java.awt.Dimension(1000, 800));
```

```
        ContenedorPaneles.setPreferredSize(new java.awt.Dimension(800, 700));
        PanelBienvenida.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        PanelBienvenida.setBorder(new
javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.RAISED));
        jLabel1.setFont(new java.awt.Font("Century Gothic", 1, 24));
        jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        jLabel1.setText("COMERCIO ELECTR\u00d3NICO 1.0");
        jLabel1.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        PanelBienvenida.add(jLabel1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(160, 250, -1, -1));

        jLabel2.setFont(new java.awt.Font("Century Gothic", 0, 18));
        jLabel2.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        jLabel2.setText("La mejor herramienta de comercio electr\u00f3nico en Internet");
        PanelBienvenida.add(jLabel2, new
org.netbeans.lib.awtextra.AbsoluteConstraints(80, 310, -1, -1));

        jLabel3.setFont(new java.awt.Font("Century Gothic", 0, 18));
        jLabel3.setText("Bienvenido a");
        PanelBienvenida.add(jLabel3, new
org.netbeans.lib.awtextra.AbsoluteConstraints(260, 220, -1, -1));

        ContenedorPaneles.addTab("BIENVENIDA", PanelBienvenida);

        PanelAnunciar.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        PanelAnunciar.setBorder(new
javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.RAISED));
        PanelDatosVendedor.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        PanelDatosVendedor.setBorder(new javax.swing.border.TitledBorder(null, "Datos
del Vendedor", javax.swing.border.TitledBorder.DEFAULT_JUSTIFICATION,
javax.swing.border.TitledBorder.DEFAULT_POSITION, new java.awt.Font("MS Sans
Serif", 1, 12)));
        PanelDatosVendedor.add(CampoLocalidad, new
org.netbeans.lib.awtextra.AbsoluteConstraints(110, 200, 210, -1));

        PanelDatosVendedor.add(CampoDNI, new
org.netbeans.lib.awtextra.AbsoluteConstraints(110, 80, 80, -1));

        EtCIF.setText("CIF");
        PanelDatosVendedor.add(EtCIF, new
org.netbeans.lib.awtextra.AbsoluteConstraints(330, 120, -1, -1));

        PanelDatosVendedor.add(CampoNombre, new
org.netbeans.lib.awtextra.AbsoluteConstraints(110, 40, 110, -1));
        CampoNombre.getAccessibleContext().setAccessibleName("");

        EtTelefono.setText("Tel\u00e9fono");
```

```
        PanelDatosVendedor.add(EtTelefono, new
org.netbeans.lib.awtextra.AbsoluteConstraints(240, 80, -1, -1));

        PanelDatosVendedor.add(CampoCP, new
org.netbeans.lib.awtextra.AbsoluteConstraints(360, 160, 70, -1));

        PanelDatosVendedor.add(CampoCIF, new
org.netbeans.lib.awtextra.AbsoluteConstraints(360, 120, 70, -1));

        EtEmpresa.setText("Empresa");
        PanelDatosVendedor.add(EtEmpresa, new
org.netbeans.lib.awtextra.AbsoluteConstraints(50, 120, -1, -1));

        PanelDatosVendedor.add(CampoDireccion, new
org.netbeans.lib.awtextra.AbsoluteConstraints(110, 160, 210, -1));

        EtDNI.setText("DNI");
        PanelDatosVendedor.add(EtDNI, new
org.netbeans.lib.awtextra.AbsoluteConstraints(50, 80, -1, -1));

        PanelDatosVendedor.add(CampoProvincia, new
org.netbeans.lib.awtextra.AbsoluteConstraints(390, 200, 110, -1));

        EtCP.setText("C.P.");
        PanelDatosVendedor.add(EtCP, new
org.netbeans.lib.awtextra.AbsoluteConstraints(330, 160, -1, -1));

        PanelDatosVendedor.add(CampoTelefono, new
org.netbeans.lib.awtextra.AbsoluteConstraints(300, 80, 80, -1));

        EtDireccion.setText("Direccion");
        PanelDatosVendedor.add(EtDireccion, new
org.netbeans.lib.awtextra.AbsoluteConstraints(50, 160, -1, -1));

        PanelDatosVendedor.add(CampoApellidos, new
org.netbeans.lib.awtextra.AbsoluteConstraints(300, 40, 190, -1));

        EtApellidos.setText("Apellidos");
        PanelDatosVendedor.add(EtApellidos, new
org.netbeans.lib.awtextra.AbsoluteConstraints(240, 40, -1, -1));

        EtPoblacion.setText("Localidad");
        PanelDatosVendedor.add(EtPoblacion, new
org.netbeans.lib.awtextra.AbsoluteConstraints(50, 200, -1, -1));

        PanelDatosVendedor.add(CampoEmpresa, new
org.netbeans.lib.awtextra.AbsoluteConstraints(110, 120, 150, -1));

        EtProvincia.setText("Provincia");
        PanelDatosVendedor.add(EtProvincia, new
```

```
org.netbeans.lib.awtextra.AbsoluteConstraints(330, 200, -1, -1));

        EtNombre.setText("Nombre");
        EtNombre.setAlignmentX(2.0F);
        EtNombre.setAlignmentY(2.0F);
        PanelDatosVendedor.add(EtNombre, new
org.netbeans.lib.awtextra.AbsoluteConstraints(50, 40, -1, -1));

        PanelAnunciar.add(PanelDatosVendedor, new
org.netbeans.lib.awtextra.AbsoluteConstraints(40, 20, 540, 250));

        PanelDatosProducto.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        PanelDatosProducto.setBorder(new javax.swing.border.TitledBorder(null, "Datos
del Producto", javax.swing.border.TitledBorder.DEFAULT_JUSTIFICATION,
javax.swing.border.TitledBorder.DEFAULT_POSITION, new java.awt.Font("MS Sans
Serif", 1, 12)));
        EtCodProducto.setText("Codigo del Producto");
        PanelDatosProducto.add(EtCodProducto, new
org.netbeans.lib.awtextra.AbsoluteConstraints(50, 50, -1, -1));

        EtCategoria.setText("Categoria");
        PanelDatosProducto.add(EtCategoria, new
org.netbeans.lib.awtextra.AbsoluteConstraints(50, 90, -1, -1));

        EtSubcategoria.setText("Subcategoria");
        PanelDatosProducto.add(EtSubcategoria, new
org.netbeans.lib.awtextra.AbsoluteConstraints(310, 90, -1, -1));

        EtProducto.setText("Producto (*)");
        PanelDatosProducto.add(EtProducto, new
org.netbeans.lib.awtextra.AbsoluteConstraints(50, 130, -1, -1));

        EtDescripcion.setText("Descripci\u00f3n");
        PanelDatosProducto.add(EtDescripcion, new
org.netbeans.lib.awtextra.AbsoluteConstraints(50, 170, -1, -1));

        CampoCodProducto.setEditable(false);
        PanelDatosProducto.add(CampoCodProducto, new
org.netbeans.lib.awtextra.AbsoluteConstraints(170, 50, 90, -1));

        PanelDatosProducto.add(CampoNombreProducto, new
org.netbeans.lib.awtextra.AbsoluteConstraints(120, 130, 140, -1));

        EtPrecio.setText("Precio");
        PanelDatosProducto.add(EtPrecio, new
org.netbeans.lib.awtextra.AbsoluteConstraints(310, 130, -1, -1));

        PanelDatosProducto.add(CampoPrecio, new
org.netbeans.lib.awtextra.AbsoluteConstraints(370, 130, 80, -1));
```

```
        EtEuro.setText("\u20ac");
        PanelDatosProducto.add(EtEuro, new
org.netbeans.lib.awtextra.AbsoluteConstraints(460, 130, 20, 20));

        try{

            String
cadenacat[]=ObjCategorias.ExtraeContenidoElementos(ObjCategorias.categorias);
            CampoOpcCategoria.setModel(new
javax.swing.DefaultComboBoxModel(cadenacat));

        }catch (Exception e) {

            System.out.println("Error interno " + e);

        }
        CampoOpcCategoria.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                CampoOpcCategoriaActionPerformed(evt);
            }
        });

        PanelDatosProducto.add(CampoOpcCategoria, new
org.netbeans.lib.awtextra.AbsoluteConstraints(120, 90, 130, -1));

        try{
            int indiceCatSelec=CampoOpcCategoria.getSelectedIndex();
            Element
subcategorias[]=ObjCategorias.ExtraeSubelementos(ObjCategorias.categorias[indiceCa
tSelec]);
            String
cadenasubcat[]=ObjCategorias.ExtraeContenidoElementos(subcategorias);
            CampoOpcSubcategoria.setModel(new
javax.swing.DefaultComboBoxModel(cadenasubcat));

        }catch (Exception e) {

            System.out.println("Error interno " + e);

        }
        PanelDatosProducto.add(CampoOpcSubcategoria, new
org.netbeans.lib.awtextra.AbsoluteConstraints(390, 90, 120, -1));

        TextoDescripcion.setBorder(new
javax.swing.border.EtchedBorder(javax.swing.border.EtchedBorder.RAISED));
        PanelDatosProducto.add(TextoDescripcion, new
org.netbeans.lib.awtextra.AbsoluteConstraints(120, 170, 140, 70));

        BotonAnunciar.setFont(new java.awt.Font("MS Sans Serif", 1, 11));
```

```
        BotonAnunciar.setText("ANUNCIAR");
        BotonAnunciar.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                BotonAnunciarActionPerformed(evt);
            }
        });

        PanelDatosProducto.add(BotonAnunciar, new
org.netbeans.lib.awtextra.AbsoluteConstraints(360, 190, 100, 40));

        PanelDatosProducto.add(CampoFechaAnuncio, new
org.netbeans.lib.awtextra.AbsoluteConstraints(420, 50, 90, -1));

        EtFechaAnuncio.setText("Fecha del Anuncio");
        PanelDatosProducto.add(EtFechaAnuncio, new
org.netbeans.lib.awtextra.AbsoluteConstraints(310, 50, -1, -1));

        PanelAnunciar.add(PanelDatosProducto, new
org.netbeans.lib.awtextra.AbsoluteConstraints(40, 370, 540, 260));

        PanelDatosRed.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        PanelDatosRed.setBorder(new javax.swing.border.TitledBorder(null, "Datos de la
red \"Comercio Electr\u00f3nico 1.0\"",
javax.swing.border.TitledBorder.DEFAULT_JUSTIFICATION,
javax.swing.border.TitledBorder.DEFAULT_POSITION, new java.awt.Font("MS Sans
Serif", 1, 12)));
        EtNombreDominio.setText("Nombre de Dominio:");
        PanelDatosRed.add(EtNombreDominio, new
org.netbeans.lib.awtextra.AbsoluteConstraints(40, 40, -1, -1));

        EtIdentidadDominio.setText("Identidad de Dominio:");
        PanelDatosRed.add(EtIdentidadDominio, new
org.netbeans.lib.awtextra.AbsoluteConstraints(280, 40, -1, -1));

        PanelDatosRed.add(CampoNombreDominio, new
org.netbeans.lib.awtextra.AbsoluteConstraints(160, 40, 100, -1));

        PanelDatosRed.add(CampoIdentidadDominio, new
org.netbeans.lib.awtextra.AbsoluteConstraints(410, 40, 100, -1));

        PanelAnunciar.add(PanelDatosRed, new
org.netbeans.lib.awtextra.AbsoluteConstraints(40, 280, 540, 80));

        jLabel4.setForeground(new java.awt.Color(255, 0, 0));
        jLabel4.setText("(*) Campo Obligatorio");
        PanelAnunciar.add(jLabel4, new
org.netbeans.lib.awtextra.AbsoluteConstraints(50, 640, -1, -1));

        ContenedorPaneles.addTab("ANUNCIAR", PanelAnunciar);
```

```
        PanelBuscar.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        PanelBuscar.setBorder(new
javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.RAISED));
        BotonBuscar.setText("BUSCAR");
        BotonBuscar.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                BotonBuscarActionPerformed(evt);
            }
        });

        PanelBuscar.add(BotonBuscar, new
org.netbeans.lib.awtextra.AbsoluteConstraints(360, 40, -1, -1));

        EtProdEncontrados.setText("PRODUCTOS ENCONTRADOS");
        PanelBuscar.add(EtProdEncontrados, new
org.netbeans.lib.awtextra.AbsoluteConstraints(30, 90, -1, -1));

        TablaProdEncontrados.setBorder(new
javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.RAISED));
        TablaProdEncontrados.setModel(modeloEncontrados);
        TablaProdEncontrados.setToolTipText("");

TablaProdEncontrados.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_ALL
_COLUMNS);
        TablaProdEncontrados.setName("");
        TablaProdEncontrados.setShowVerticalLines(false);

TablaProdEncontrados.setDebugGraphicsOptions(javax.swing.DebugGraphics.NONE_
OPTION);
        TablaProdEncontrados.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                MostrarProductosEncontrados(evt);
            }
        });

        jScrollPane1.setViewportView(TablaProdEncontrados);

        PanelBuscar.add(jScrollPane1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(30, 120, 560, 530));

        PanelBuscar.add(CamProdBusc, new
org.netbeans.lib.awtextra.AbsoluteConstraints(190, 40, 110, -1));

        EtProdBusc.setText("Producto Buscado:");
        PanelBuscar.add(EtProdBusc, new
org.netbeans.lib.awtextra.AbsoluteConstraints(50, 40, 120, -1));

        ContenedorPaneles.addTab("BUSCAR", PanelBuscar);
```

```
        PanelCompras.setLayout(new java.awt.BorderLayout());

        PanelCompras.setBorder(new
javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.RAISED));
        TablaProdComprados.setModel(modeloComprados);
        TablaProdComprados.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                MostrarProductoComprado(evt);
            }
        });

        jScrollPane3.setViewportView(TablaProdComprados);

        PanelCompras.add(jScrollPane3, java.awt.BorderLayout.CENTER);

        ContenedorPaneles.addTab("COMPRAS", PanelCompras);

        PanelVentas.setLayout(new java.awt.BorderLayout());

        PanelVentas.setBorder(new
javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.RAISED));
        TablaProdVendidos.setModel(modeloVendidos);
        TablaProdVendidos.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                MostrarProductoVendido(evt);
            }
        });

        jScrollPane4.setViewportView(TablaProdVendidos);

        PanelVentas.add(jScrollPane4, java.awt.BorderLayout.CENTER);

        ContenedorPaneles.addTab("VENTAS", PanelVentas);

        PanelProdAnunciados.setLayout(new java.awt.BorderLayout());

        PanelProdAnunciados.setBorder(new
javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.RAISED));
        TablaProdAnunciados.setModel(modeloAnunciados);
        TablaProdAnunciados.setShowVerticalLines(false);
        TablaProdAnunciados.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                MostrarProductoAnunciado(evt);
            }
        });

        jScrollPane2.setViewportView(TablaProdAnunciados);

        PanelProdAnunciados.add(jScrollPane2, java.awt.BorderLayout.CENTER);
```

```
        ContenedorPaneles.addTab("PRODUCTOS ANUNCIADOS",
PanelProdAnunciados);

        getContentPane().add(ContenedorPaneles, java.awt.BorderLayout.CENTER);

        pack();
    }

//    Método que es llamado cuando se selecciona una fila de la tabla del
//    panel de productos vendidos y que se encarga de mostrar en detalle el
//    fichero XML del producto seleccionado.
    private void MostrarProductoVendido(java.awt.event.MouseEvent evt) {

        int filaSeleccionada=TablaProdVendidos.getSelectedRow();

        ProductoXML
productoSeleccionado=(ProductoXML)modeloVendidos.listaProductos.get(filaSeleccio
nada);

        MuestraProducto panel=new MuestraProducto(productoSeleccionado);
        panel.show();
    }

//    Método que es llamado cuando se selecciona una fila de la tabla del
//    panel de productos comprados y que se encarga de mostrar en detalle el
//    fichero XML del producto seleccionado.
    private void MostrarProductoComprado(java.awt.event.MouseEvent evt) {

        int filaSeleccionada=TablaProdComprados.getSelectedRow();

        ProductoXML
productoSeleccionado=(ProductoXML)modeloComprados.listaProductos.get(filaSelecc
ionada);

        MuestraProducto panel=new MuestraProducto(productoSeleccionado);
        panel.show();
    }

//    Método que es llamado cuando se selecciona una fila de la tabla del
//    panel de productos encontrados y que se encarga de mostrar en detalle el
//    fichero XML del producto seleccionado.
    private void MostrarProductosEncontrados(java.awt.event.MouseEvent evt) {

        int filaSeleccionada=TablaProdEncontrados.getSelectedRow();

        ProductoXML
productoSeleccionado=(ProductoXML)modeloEncontrados.listaProductos.get(filaSelec
cionada);
        TuberiaCliente tuberia=(TuberiaCliente)cliente.listaTuberias.get(filaSeleccionada);
```

```
        tuberia.filaSeleccionada=filaSeleccionada;

        MuestraProductoEncontrado panel=new
MuestraProductoEncontrado(productoSeleccionado,tuberia,modeloComprados);
        panel.show();
    }

//    Método que es llamado cuando se selecciona una fila de la tabla del
//    panel de productos anunciados y que se encarga de mostrar en detalle el
//    fichero XML del producto seleccionado.
    private void MostrarProductoAnunciado(java.awt.event.MouseEvent evt) {

        int filaSeleccionada=TablaProdAnunciados.getSelectedRow();

        ProductoXML
productoSeleccionado=(ProductoXML)modeloAnunciados.listaProductos.get(filaSelecc
ionada);

        MuestraProducto panel=new MuestraProducto(productoSeleccionado);
        panel.show();
    }

//    Método que es llamado cuando se pulsa el botón "Buscar" del panel con el
//    mismo nombre, y que se encarga de instanciar la clase Buscar que será
//    la responsable de buscar el servicio JXTA cuyo nombre coincida con la
//    cadena introducida por el usuario en el campo dedicado a tal fin
//    de dicho panel
    private void BotonBuscarActionPerformed(java.awt.event.ActionEvent evt) {

        String palabraBusqueda="*"+CamProdBusc.getText()+"*";

        Buscar busqueda=new Buscar();
        busqueda.BuscarProducto(palabraBusqueda, plataforma, modeloEncontrados);
        cliente=busqueda.cliente;
    }

//    Método llamado al pulsar el botón "Anunciar" del panel con el mismo
//    nombre y que se encarga de recoger toda la información introducida por el
//    usuario en dicho panel e instanciar un objeto de la clase ProductoXML.
//    Con dicho objeto se instancia otro de la clase Anunciar que será el
//    responsable de anunciar dicho servicio en la red.
    private void BotonAnunciarActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        String nombre=CampoNombre.getText();
        String apellidos=CampoApellidos.getText();
        String direccion=CampoDireccion.getText();
        String localidad=CampoLocalidad.getText();
        String provincia=CampoProvincia.getText();
        String cp=CampoCP.getText();
        String telefono=CampoTelefono.getText();
```

```
        String dni=CampoDNI.getText();
        String cif=CampoCIF.getText();
        String empresa=CampoEmpresa.getText();

        String codigoProd=CampoCodProducto.getText();
        String categoria=(String)CampoOpcCategoria.getSelectedItem();
        String subcategoria=(String)CampoOpcSubcategoria.getSelectedItem();
        String nombreProducto=CampoNombreProducto.getText();
        String descripcion=TextoDescripcion.getText();
        String precio=CampoPrecio.getText();
        String fechaAnuncio=CampoFechaAnuncio.getText();

        String dominio=CampoNombreDominio.getText();
        String identidad=CampoIdentidadDominio.getText();


        ProductoXML producto=new ProductoXML();
        producto.RellenarDatosProveedor(nombre,apellidos,direccion, localidad,
provincia, cp, telefono, dni, cif, empresa);

producto.RellenarDatosProducto(codigoProd,categoria,subcategoria,nombreProducto,
descripcion, precio,fechaAnuncio);
        producto.RellenarDatosRed(identidad,dominio);

        String nombreFichero=new String(codigoProd+".xml");
        producto.guardarEnArchivo(new File(rutaProductosAnunciados+nombreFichero));


        Anunciar anuncio=new Anunciar(producto,nombreFichero,plataforma,
modeloAnunciados,modeloVendidos,listaAnuncios);

        anuncio.AnunciarProducto();
        anuncio.BorraPanelAnunciar(this);

    }

//    Método que extrae del fichero "categorias.xml" las subcategorías
//    correspondientes a una categoría seleccionada
    private void CampoOpcCategoriaActionPerformed(java.awt.event.ActionEvent evt) {

        try{
            int indiceCatSelec=CampoOpcCategoria.getSelectedIndex();
            Element
subcategorias[]=ObjCategorias.ExtraeSubelementos(ObjCategorias.categorias[indiceCa
tSelec]);
            String
cadenasubcat[]=ObjCategorias.ExtraeContenidoElementos(subcategorias);
            CampoOpcSubcategoria.setModel(new
javax.swing.DefaultComboBoxModel(cadenasubcat));
            PanelDatosProducto.add(CampoOpcSubcategoria, new
```

```
org.netbeans.lib.awtextra.AbsoluteConstraints(390, 90, 120, -1));

        }catch (Exception e) {

            System.out.println("Error interno " + e);

        }
    }

//    Método con el que se sale de la aplicación
    private void exitForm(java.awt.event.WindowEvent evt) {

        System.exit(0);
    }

//    Método principal de la aplicación cuya unica función es llamar al método
//    show() de la clase Comercio que es subclase de JFrame.
    public static void main(String args[]) {
        new Comercio().show();
    }

//    Método que determina cada vez que se ejecuta la aplicación el código del
//    nuevo producto que se va a anunciar. Para ello examina los ficheros
//    existentes en los directorios "Anuncios" y "Vendidos" y determina cuál
//    es el código más alto de ambos directorios, de forma que incrementándolo
//    en 1 sea el código asignado al nuevo producto.
     int DeterminaCodigoProducto(ModeloTablaAnunciados modeloAnunciados,
ModeloTablaVendidos modeloVendidos){

        Integer max=new Integer(0);
        Integer min;
        int comparacion;

        for(int i=0;i<modeloAnunciados.listaProductos.size();i++){

            ProductoXML prod=(ProductoXML)(modeloAnunciados.listaProductos.get(i));
            min=new Integer(prod.codigoProd.getText());
            comparacion=min.compareTo(max);
            if(comparacion>0){
                max=min;
            }

        }
        for(int i=0;i<modeloVendidos.listaProductos.size();i++){

            ProductoXML prod=(ProductoXML)(modeloVendidos.listaProductos.get(i));
            min=new Integer(prod.codigoProd.getText());
            comparacion=min.compareTo(max);
            if(comparacion>0){
                max=min;
```

```
        }

    }
    return max.intValue()+1;
}

// Variables declaration - do not modify
private javax.swing.JButton BotonAnunciar;
private javax.swing.JButton BotonBuscar;
private javax.swing.JTextField CamProdBusc;
public javax.swing.JTextField CampoApellidos;
public javax.swing.JTextField CampoCIF;
public javax.swing.JTextField CampoCP;
public javax.swing.JTextField CampoCodProducto;
public javax.swing.JTextField CampoDNI;
public javax.swing.JTextField CampoDireccion;
public javax.swing.JTextField CampoEmpresa;
public javax.swing.JTextField CampoFechaAnuncio;
private javax.swing.JTextField CampoIdentidadDominio;
public javax.swing.JTextField CampoLocalidad;
public javax.swing.JTextField CampoNombre;
private javax.swing.JTextField CampoNombreDominio;
public javax.swing.JTextField CampoNombreProducto;
private javax.swing.JComboBox CampoOpcCategoria;
private javax.swing.JComboBox CampoOpcSubcategoria;
public javax.swing.JTextField CampoPrecio;
public javax.swing.JTextField CampoProvincia;
public javax.swing.JTextField CampoTelefono;
private javax.swing.JTabbedPane ContenedorPaneles;
private javax.swing.JLabel EtApellidos;
private javax.swing.JLabel EtCIF;
private javax.swing.JLabel EtCP;
private javax.swing.JLabel EtCategoria;
private javax.swing.JLabel EtCodProducto;
private javax.swing.JLabel EtDNI;
private javax.swing.JLabel EtDescripcion;
private javax.swing.JLabel EtDireccion;
private javax.swing.JLabel EtEmpresa;
private javax.swing.JLabel EtEuro;
private javax.swing.JLabel EtFechaAnuncio;
private javax.swing.JLabel EtIdentidadDominio;
private javax.swing.JLabel EtNombre;
private javax.swing.JLabel EtNombreDominio;
private javax.swing.JLabel EtPoblacion;
private javax.swing.JLabel EtPrecio;
private javax.swing.JLabel EtProdBusc;
private javax.swing.JLabel EtProdEncontrados;
private javax.swing.JLabel EtProducto;
private javax.swing.JLabel EtProvincia;
private javax.swing.JLabel EtSubcategoria;
```

```
    private javax.swing.JLabel EtTelefono;
    private javax.swing.JPanel PanelAnunciar;
    private javax.swing.JPanel PanelBienvenida;
    private javax.swing.JPanel PanelBuscar;
    private javax.swing.JPanel PanelCompras;
    private javax.swing.JPanel PanelDatosProducto;
    private javax.swing.JPanel PanelDatosRed;
    private javax.swing.JPanel PanelDatosVendedor;
    private javax.swing.JPanel PanelProdAnunciados;
    private javax.swing.JPanel PanelVentas;
    private javax.swing.JTable TablaProdAnunciados;
    private javax.swing.JTable TablaProdComprados;
    private javax.swing.JTable TablaProdEncontrados;
    private javax.swing.JTable TablaProdVendidos;
    public javax.swing.JTextPane TextoDescripcion;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JScrollPane jScrollPane3;
    private javax.swing.JScrollPane jScrollPane4;
    // End of variables declaration

}
```

Con esta clase se concluye este anexo dedicado al código fuente de las clases que componen la aplicación desarrollada en este Proyecto Fin de Carrera