ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

DISEÑO Y DESARROLLO DE UNA APLICACIÓN CLIENTE-SERVIDOR USANDO COMO MÉTODO DE TRANSFERENCIA DE DATOS CLIENTES DE CORREO ELECTRÓNICO.

AUTOR: José Carlos Casas González.
DIRECTORES: Alejandro Santos Martínez Sala
Pilar Manzanares López.

Septiembre / 2008

Autor:	José Carlos Casas González
Email del autor:	josecarloscg@*Yahoo*.es
Director:	Alejandro Santos Martínez Sala, Pilar Manzanares López
Email del Director:	alejandros.martinez@upct.es, pilar.manzanares@upct.es
Título del PFC:	Diseño y desarrollo de una aplicación cliente-servidor usando como método de transferencia de datos clientes de correo electrónico.
Resumen:	Debido a la creciente demanda de sistemas que no requieren de la presencia humana para su control, se ha desarrollado una arquitectura cliente-servidor, basada en el protocolo de transferencia de correos electrónicos, SMTP, y de descarga POP3, encargada del envío, recepción y procesado automático de datos provenientes del exterior. Todo ello garantizando un acceso rápido y seguro a través del servicio de correo electrónico de *Gmail*, proporcionado por una de las empresas punteras en el mundo de las tecnologías de la información y las comunicaciones como es Google.
Titulación:	Ingeniería Técnica de Telecomunicación
Especialidad:	Telemática
Departamento:	Departamento de Tecnologías de la Información y las Comunicaciones
Fecha de Presentación:	Septiembre 2008

ÍNDICE

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

Capítulo 1
INTRODUCCIÓN

1. Introducción.

1.1 Antecedentes.

Actualmente se está produciendo una gran revolución en el mundo laboral, más concretamente en el entorno industrial y de las tecnologías de la información y las comunicaciones, donde la presencia humana para la supervisión y control de los diferentes procesos que se llevan a cabo, se está reduciendo de forma considerable, dando lugar a una automatización de procesos.

Como se ha comentado, en esta nueva Era toma gran importancia el trabajo que pueda realizar una máquina por sí sola, sin la supervisión humana, ya que hay actividades que deben ser controladas continuamente, siendo crítico su correcto funcionamiento; es el caso de las centrales nucleares donde segundo a segundo se deben supervisar los niveles de radiación, posibles fugas y otros imprevistos. Y qué mejor manera de llevar a cabo esta tarea contando con una serie de aplicaciones informáticas que realizan este trabajo autónomamente y estén correctamente preparadas para atender cualquier situación.

A este último punto está enfocado este proyecto, donde con una simple cuenta de correo electrónico de cualquier proveedor, podemos crear y configurar una arquitectura cliente-servidor que permita adaptarse a cualquier necesidad. Todo ello, definiendo fácilmente una serie de mensajes que serán intercambiados entre nuestras aplicaciones y un servidor de correo.

Hoy día lo que se busca es que cualquier dispositivo sea lo más pequeño posible, tenga un consumo muy bajo, pueda operar en las condiciones más adversas y, por supuesto, toda la información que transmita o reciba lo haga de forma segura, además de ser intuitivo y manejado con facilidad por aquellos operarios que se encargan de su manipulación. Esto es lo que consiguen los clientes de correo usados para implementar este proyecto, ya que se aprovecha toda la infraestructura proporcionada por el operador de correo, como pueden ser lo servidores y líneas de comunicación, siendo solamente necesario recoger datos del mundo exterior, crear un mensaje de correo electrónico y enviarlo a una cuenta de correo que actuará como *servidor*; dichos mensajes estarán ahí almacenados hasta que sean descargados en un momento determinado para ser procesados.

En resumen, aprovechando todas las posibilidades que nos brindan hoy en día los proveedores de correo electrónico se pueden construir infinidad de aplicaciones que continuamente estén enviando y procesando diferentes parámetros, adaptándose de manera fácil a cualquier requerimiento.

1.2 Objetivos.

Como se ha comentado en la introducción anterior, el objetivo de este proyecto es el estudio del servicio de correo electrónico. Para ello, además de realizar un completo tutorial de este servicio, se ha implementado una arquitectura cliente-servidor donde, haciendo uso de cuentas de correo electrónico proporcionadas en este caso por el servicio de correo electrónico que ofrece Google, *Gmail*, podamos llevar a la práctica cualquier actividad de automatización de procesos.

Como se ha comentado, este proyecto está realizado con cuentas de correo de *Gmail*, lo cual no ha sido una casualidad, ya que se ha realizado un amplio estudio comparativo entre los diferentes proveedores de correo *web* gratuitos presentes en el mercado y se ha elegido el que mejor se adapta a nuestro objetivo final.

Por lo tanto, el correo electrónico será el eje principal entorno al cual gire este trabajo; el también conocido como *email* es actualmente usado por millones de personas en el mundo, que ven una forma sencilla y barata de poder comunicarse, pero la gran mayoría desconoce toda la infraestructura subyacente que se pone en marcha cuando se envía o recibe un correo electrónico.

Por ello, se van a explicar detalladamente los protocolos en los cuales se basa el *email*; se estudiará el protocolo SMTP (*Simple Mail Transfer Protocol*) que es el encargado de que cada mensaje llegue a su destino; y el protocolo POP3 (*Post Office Protocol v3*) que es el responsable de que podamos acceder a los correos electrónicos, comparándolo con el otro protocolo de descarga que existe actualmente IMAP (*Intenet Message Access Protocol*). Se estudiarán los comandos intercambiados por todas las partes, la existencia de herramientas del lenguaje de programación que permiten implementar dichos comandos, etc.

Y, por último, a pesar de que uno de los objetivos era indagar en todo el proceso que se genera en el envío y recepción de *emails*, la meta final de este proyecto es la programación en C# de las aplicaciones que permitirán visualizar y comprender mejor el funcionamiento de todo lo estudiado.

Comentar, de un modo general, que esta arquitectura, está formada por una serie de aplicaciones *Cliente* de correo electrónico encargadas de recoger y enviar datos procedentes del exterior a una aplicación *Servidor* que se encargará de procesar dichos datos y enviar la respuesta al cliente correspondiente. Todo ello se llevará a cabo usando los protocolos que implementa el servicio de correo electrónico, SMTP y POP3.

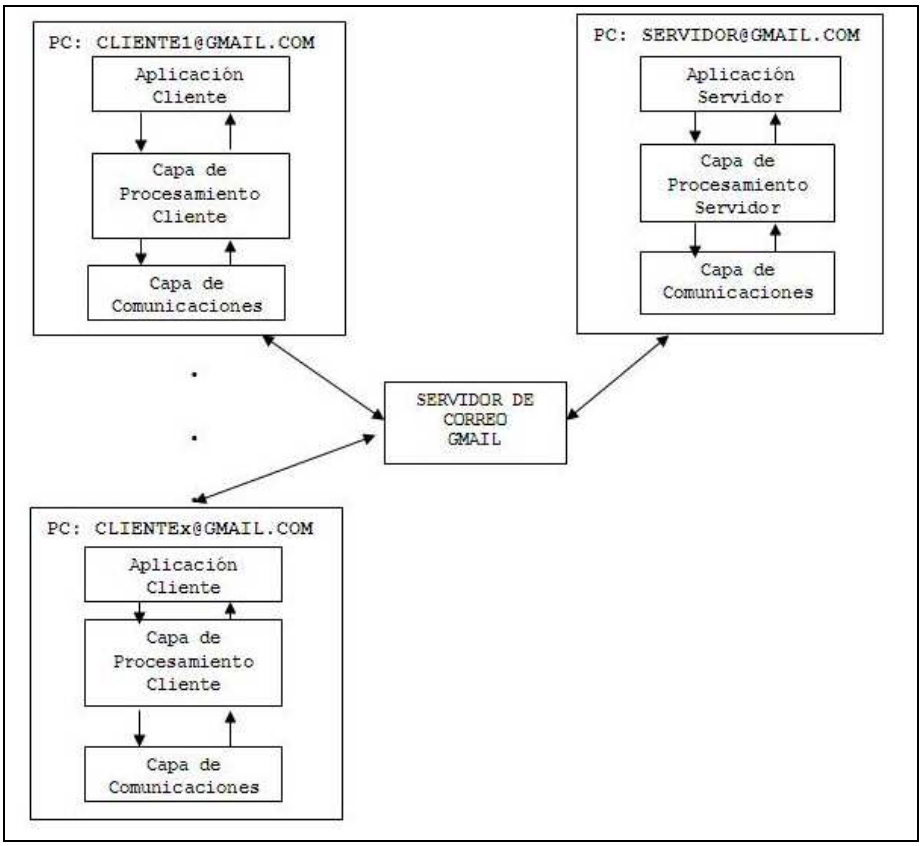


Figura 1: Esquema general de la aplicación a desarrollar.

1.3 Herramientas utilizadas en la implementación

La implementación de la arquitectura Cliente-Servidor se ha llevado a cabo gracias a la herramienta de programación Microsoft Visual C# 2005, en su edición Express (Para más información se puede consultar la referencia [4] de la bibliografía.)

Durante el desarrollo se han hecho uso de los clientes de correo *Mozilla Thunderbird 2.0.0.14* y *Microsoft Outlook 2008*, que han permitido comparar las aplicaciones implementadas con dos de las aplicaciones comerciales de correo electrónico más usadas actualmente.

El analizador de redes *Ethereal 0.99.0*, ha sido usado a lo largo de este proyecto en numerosas ocasiones para poder observar el intercambio de mensajes llevado a cabo en todo momento por las aplicaciones, lo que ha permitido verificar el correcto funcionamiento de este trabajo.

Por último, indicar que, a pesar de usar para la aplicación final cuentas de correo del proveedor *Gmail*, se han probado otras cuentas de correo como son las de *Yahoo* y *Cartagena*, sirviendo esta última para observar el funcionamiento de conexiones no seguras.

1.4 Estructura de la memoria del proyecto.

En el primer capítulo de este trabajo, se pone en antecedentes al lector sobre las líneas a seguir en el desarrollo de la arquitectura; se exponen los objetivos a alcanzar con este proyecto, así como las herramientas utilizadas para ello.

El segundo capítulo sirve realiza un profundo estudio sobre el servicio de correo electrónico, donde se analiza de forma genérica el correo electrónico; la elección de *Gmail* como proveedor de las cuentas necesarias para la implementación; además de realizar un detallado tutorial de los protocolos que intervienen en una aplicación de correo electrónico, esto es, SMTP, POP3 e IMAP. Todo ello, bajo el contexto de la seguridad, concepto muy relevante hoy en día.

El tercer capítulo, expone detalladamente, el desarrollo de las aplicaciones implementadas, todas ellas desglosadas por capas y funcionalidades. Se describe tanto la aplicación *Servidor* encargada de procesar los correos, como las aplicaciones *Cliente* encargadas de su envío y recepción.

La configuración de las aplicaciones, es descrita en el capítulo cuatro, donde se muestra como configurar las cuentas de correo así como los archivos de configuración de los programas implementados.

El capítulo cinco de esta memoria, ilustra un ejemplo demostrativo del funcionamiento de la aplicación.

Las diversas pruebas realizadas para comprobar la fiabilidad y robustez de las aplicaciones, así como las conclusiones extraídas y las líneas de trabajo futuras son expuestas en el capítulo seis.

El capítulo siete muestra la bibliografía utilizada para el desarrollo de este trabajo.

Por último, se incluye una sección de anexos con las clases más relevantes proporcionadas por la librería *System.Net.Mail* de la API de C#.

Capítulo 2
ESTUDIO DEL SERVICIO DE CORREO ELECTRÓNICO

2. Estudio del servicio de correo electrónico.

2.1 El correo electrónico.

2.1.1 ¿Qué es el correo electrónico?

Es una herramienta telemática basada en un conjunto de técnicas y servicios que combinan las telecomunicaciones y la informática y que se constituye en el correo del Tercer Milenio. Ofrece un correo sin barreras de tiempo y espacio, que viaja en fracciones de segundos, con textos, sonidos e imágenes. Además permite enviar el mensaje de correo electrónico a uno o varios remitentes al mismo tiempo, con dirección visible o encriptada, con listas de distribuciones públicas o privadas.

2.1.2 Elementos del correo electrónico.

2.1.2.1 Dirección de correo.

Una dirección de correo electrónico es un conjunto de palabras que identifican a una persona que puede enviar y recibir correo. Cada dirección es única y pertenece siempre a la misma persona (persona@servicio.com).

2.1.2.1 Proveedor de correo.

Para poder usar, enviar y recibir correo electrónico generalmente hay que estar registrado en alguna empresa que ofrezca este servicio (gratuita o de pago). El registro permite tener una *dirección de correo* personal única y duradera, a la que se puede acceder mediante un nombre de usuario y una contraseña.

Hay varios tipos de proveedores de correo, que se diferencian sobre todo por la calidad del servicio que ofrecen. Básicamente, se pueden dividir en dos tipos: los correos gratuitos y los de pago.

<u>Gratuitos</u>
Los correos gratuitos son los más usados, aunque incluyen algo de publicidad: unos incrustada en cada mensaje y otros en la interfaz que se usa para leer el correo. Muchos sólo permiten ver el correo desde una página *web* propia del proveedor, para asegurarse de que los usuarios reciben la publicidad que se encuentra ahí. En cambio, otros permiten también usar un programa de correo configurado para que se descargue el correo de forma automática.

Una desventaja de estos correos es que en cada dirección, la parte que hay a la derecha de la **@** muestra el nombre del proveedor; por ejemplo, el usuario *gapa* puede acabar teniendo *gapa@correo-gratuito.net*. Este tipo de direcciones desagradan a algunos (sobre todo, a empresas) y por eso es común comprar un dominio propio, para dar un aspecto más profesional.

De pago

Los correos de pago normalmente ofrecen todos los servicios disponibles. Es el tipo de correo que un proveedor de Internet da cuando se contrata la conexión. También es muy común que una empresa registradora de dominios venda, junto con el dominio, varias cuentas de correo para usar junto con ese dominio (normalmente, más de 1).

También se puede considerar *de pago* el método de comprar un nombre de dominio e instalar un ordenador servidor de correo con los programas apropiados (un MTA). No hay que pagar cuotas por el correo, pero sí por el dominio, y también los gastos que da mantener un ordenador encendido todo el día.

2.1.3 Ventajas y problemas del correo electrónico.

Ventajas respecto al correo tradicional:

- Al principio, el correo electrónico, solo podía enviar mensajes de texto con mayor o menor rapidez. En la actualidad es posible enviar todo tipo de datos binarios gracias a estándares como MIME o UUDECODE. Se pueden incluir como parte del mensaje imágenes, sonidos, ficheros binarios, programas ejecutables, etc. Para ello es necesario que tanto el usuario que envía el correo como el que lo recibe dispongan de un gestor de correo que cumplan o incluyan estos estándares.

- Otra característica importante del *email*, es que si un usuario tiene un acceso limitado a Internet, con su cuenta de correo puede sacar mucho provecho a servicios como el FTP, ARCHIE, LISTAS DE CORREO, etc.

- Otra de las ventajas del *email*, es que no nos tenemos que preocupar de comprobar si llegó algún tipo de correo. La máquina de la que somos usuarios, se encarga de comprobarlo por nosotros y avisarnos cuando nos conectamos a dicha máquina. En el correo tradicional, somos nosotros quienes nos encargamos de mirar en el buzón para comprobar si ha llegado algo.

Entre los *problemas* actuales que puede encontrar el correo electrónico, se encuentran los siguientes:

- El *spam*, que se refiere a la recepción de correos no solicitados, normalmente de publicidad engañosa, y en grandes cantidades.

- Los *virus informáticos*, que se propagan mediante ficheros adjuntos infectando el ordenador de quien los abre.

- El *phishing*, que son correos fraudulentos que intentan conseguir información bancaria.

- Los *engaños (hoax),* que difunden noticias falsas masivamente mediante las *cadenas de correo electrónico*, que consisten en reenviar un mensaje a mucha gente; aunque parece inofensivo, la publicación de listas de direcciones de correo contribuye a la propagación a gran escala del *spam* y de mensajes con virus, *phishing* y *hoax*.

2.1.4 Correos Web y Clientes de correo.

2.1.4.1 Correo *Web*.

Permiten enviar y recibir correos mediante una página *web* diseñada para ello, y por tanto usando sólo un programa navegador *web*.

El *correo web* es cómodo para mucha gente, porque permite ver y almacenar los mensajes desde cualquier sitio (en un servidor remoto, accesible por la página *web*) en vez de en un ordenador personal concreto.

Como desventaja, es difícil de ampliar con otras funcionalidades, porque la página ofrece unos servicios concretos y no podemos cambiarlos. Además, suele ser más lento que un *programa de correo*, ya que hay que estar continuamente conectado a páginas *web* y leer los correos de uno en uno.

2.1.4.2 Clientes de correo.

Programas para gestionar los mensajes recibidos y poder escribir nuevos.

Incorporan muchas más funcionalidades que el *correo web*, ya que todo el control del correo pasa a estar en el ordenador del usuario. Por ejemplo, algunos incorporan potentes filtros anti-spam.

Por el contrario, necesitan que el proveedor de correo ofrezca este servicio, ya que no todos permiten usar un programa especializado (algunos sólo dan *correo web*). En caso de que sí lo permita, el proveedor tiene que explicar detalladamente cómo hay que configurar el programa de correo. Esta información siempre está en su página *web*, ya que es imprescindible para poder hacer funcionar el programa, y es distinta en cada proveedor. Entre los datos necesarios están: tipo de conexión (POP o IMAP), *dirección del servidor de correo*, *nombre de usuario* y *contraseña*. Con estos datos, el programa ya es capaz de obtener y descargar nuestro correo.

El funcionamiento de un *programa de correo* es muy diferente al de un *correo web*, ya que un programa de correo descarga de golpe *todos* los mensajes que tenemos disponibles, y luego pueden ser leídos sin estar conectados a Internet (además, se quedan grabados en el ordenador). En cambio, en una página *web* se leen de uno en uno, y hay que estar conectado a la red todo el tiempo.
Algunos ejemplos de programas de correo son Mozilla Thunderbird, Outlook Express y Eudora.

2.1.4.3 ¿Cuál utilizar?

La utilización de una u otra forma de acceder al correo, depende de las necesidades de cada usuario. Entre los factores más decisivos se encuentran los siguientes:

- La movilidad: al usuario que se desplace con frecuencia o que acceda habitualmente al correo electrónico desde más de un ordenador, le será más cómodo utilizar un correo *web* en lugar de tener los mensajes repartidos entre varias computadoras.

- Si no se dispone de alta velocidad (ADSL o cable) o al menos de tarifa plana, es más económico emplear una cuenta POP, pues sólo hace falta conectarse para enviar y recibir correo y se puede escribir o leer los mensajes fácilmente sin estar conectado.

- Se supone que un gestor de correo es más eficaz para filtrar y organizar el correo en diferentes carpetas, localizar los mensajes o añadir programas para evitar los virus y el *spam* (mensajes comerciales no deseados). Sin embargo, las cuentas de correo *web* actuales no sólo han aumentado su capacidad, casi todas incluyen potentes buscadores y la posibilidad de crear carpetas y reglas de mensaje, así como filtros anti-spam y anti-virus.

- Al configurar una cuenta en un programa de correo es posible utilizar el protocolo HTTP en lugar de POP. Es decir, se puede configurar una cuenta de correo-*web* (las que lo permitan) para recibirla directamente al PC. También hay algunos correo *web* (como *Gmail* y *Yahoo*!) que permiten configurar un acceso POP de forma gratuita.

- Asimismo, existe un híbrido entre los clientes de correo y los correos *web*, donde muchos proveedores ofrecen la posibilidad de acceder a las cuentas POP a través de una página *web*, donde podremos ver los mensajes que todavía no hemos descargado al ordenador. Así, también es posible acceder a una cuenta POP mediante el navegador desde cualquier ordenador conectado.

2.1.5 Comparativa de los diferentes correos Web gratuitos más destacados actualmente.

Para realizar la comparativa de los correos *web*s actuales, este trabajo se ha basado en un estudio realizado por consumer.es EROSKI. Este documento solo va a analizar tres de los correos *web*s ofertados (*Gmail, Yahoo* y *Hotmail*), por ser los que ofrecen unas mayores prestaciones a los usuarios.

2.1.5.1 Tabla comparativa.

	GMAIL	*HOTMAIL*	*YAHOO*
ESPACIO	2.65GB	2GB	1GB
NAVEGADORES COMPATIBLES	IE 5.5, Firefox, Opera, Netscape, Safari	IE 6; sólo características básicas en otros navegadores	IE, Firefox, Opera
INTEGRACIÓN CON OTROS SERVICIOS	Integración con Google Calendar beta, Google Talk (Chat) beta; localizaciones enlazan a Google Maps	Muestra Windows Live Calendar beta	Enlaza a*Yahoo* Calendar y Notepad
POSIBLIDAD DE "DRAG AND DROP"	No	Si	Si
ORGANIZACIÓN DE LOS MENSAJES	Etiquetas	Carpetas	Carpetas
SEGURIDAD	Filtros de Spam y Virus (No acepta ejecutables)	Filtros de Spam(3 niveles),Pishing y Virus (Analiza al enviar y a descargar)	Filtros de Spam y Virus (Analiza al enviar y a descargar)
BÚSQUEDA DE MENSAJES	También en archivos adjuntos	Si pero sólo mensajes	También en archivos adjuntos
PUBLICIDAD	En la cabecera y dentro del contenido del email, sólo texto	Anuncios animados y con imágenes	Anuncios animados y con imágenes
LECTOR RSS	Implementado en el interfaz	No	Si
ALMACENAMIENTO DE MENSAJES ENVIDOS	Si	Si	Si
ESTILO CONFIGURABLE	No	9 temas de colores	Con templates
ATAJOS DE TECLADO	Si	Si, botón derecho en los menus	Si, botón derecho en los menus
FILTROS PARA MENSAJE	Si	Si	Si
CORRECTOR ORTOGRÁFICO	Sí,en 30 idiomas	Sólo en IE y con posibles errores	Solo en IE
SOPORTE	Foros y por medio de búsquedas	Sección de búsqueda y FAQ	Tutoriales animados
INTERFAZ ALTERNATIVA	Interfaz HTML	Interfaz *Hotmail* clásico	Interfaz *Yahoo* clásico
TAMAÑO ARCHIVOS ADJUNTOS	ilimitados 10MB	ilimitados 10MB	5 archivos 10 MB

CONSULTA CORREO EXTERNO	No	De pago	Sí
ACCESO POP	Sí	De pago	con suscripción a boletines
REENVÍO AUTOMÁTICO	Sí	No	con suscripción a boletines
BAJA POR INACTIVIDAD	9 meses	30 días mensajes 90 días cuenta	a discreción
CONTACTOS	frecuentes importar	grupos favoritos importar	grupos carpetas / rápidos importar exportar

Tabla 1: Comparativa de los diferentes correos *web* gratuitos disponibles actualmente.

2.1.5.2 Conclusiones.

Existen enormes diferencias entre los correo-*web* en cuanto a capacidad: desde los 25 MB de *Hotmail*, ampliables al cabo de un mes de uso a 250 MB, "a discreción de Microsoft", hasta los más de 2.200 MB de *Gmail*.

Salvo en *Gmail* es necesario pagar, *Hotmail*, o recibir envíos promocionales ,*Yahoo*, para recibir los mensajes en el ordenador a través de un programa gestor de correo como Outlook Express, Eudora o Thunderbird (lo que se conoce como 'acceso POP').

La seguridad es el único aspecto que merece un suspenso casi general: sólo *Gmail*, *Yahoo*! y *Hotmail* ofrecen una conexión segura para acceder a las cuentas.

El nivel de las prestaciones ofrecidas es en general alto, con numerosas herramientas para organizar los mensajes y los contactos. Únicamente en *Gmail* y *Yahoo*! es posible encontrar emails buscando en el texto del mensaje.

Dificultades de uso tanto en Mozilla Firefox como en Internet Explorer (IE), *Hotmail* y *Yahoo*! cuentan con herramientas (como el editor de HTML) que sólo funcionan en el navegador de Microsoft.

Sólo *Yahoo*! y *Hotmail* afirman analizar en busca de virus los archivos adjuntos recibidos y enviados, aunque *Gmail*, es también bastante eficaz a la hora de cerrar las puertas a los virus aunque no lo afirme en sus páginas: el 48,5% de los virus enviados no alcanzó su destino y el 34,07% fue tratado como 'spam'. Sólo *Yahoo*!, *Hotmail* y *Gmail* impidieron que se adjuntaran o enviaran virus desde sus buzones.

El proceso de Alta es en general rápido y sencillo. Únicamente *Gmail* restringe el acceso a su servicio (es necesario recibir una invitación, aunque sencilla de conseguir), a cambio, solicita los datos imprescindibles. Por el contrario, *Hotmail* exigen hasta 11 y 20 datos personales, respectivamente, para formalizar el alta.

La mayoría de los mensajes se entregó con gran velocidad. Sin embargo, *Hotmail* se mostró lento a la hora de recibir archivos pesados.
El tamaño de los archivos adjuntos permitidos (tanto para enviar como para recibir) es de 10 MB. *Gmail* no permite por seguridad el envío o recepción de archivos ejecutables (.exe), y en ocasiones falla en la recepción de archivos comprimidos (.zip) aunque no contengan ejecutables.

Ninguno de los correo-*web* dispone de una versión adaptada para personas discapacitadas, ni se encuentra mención alguna al respecto en las páginas de ayuda.

2.1.5.3 Motivos de la elección de *Gmail*.

- Gran capacidad.

- Búsqueda de mensajes sencilla, mediante un sistema de "etiquetas" personalizadas, en vez de carpetas. Eso hace que un solo mensaje pueda estar en distintas categorías (etiquetas), en vez de montar un árbol de directorios. Esta versatilidad es impensable con un sistema de carpetas.

- Búsqueda integrada. Cuando hay una gran cantidad de mensajes, la búsqueda se puede realizar al igual que la localización de una *web* en Google.

- Mensajes contextualizados en conversaciones. El envío de correos consecutivos con un mismo contacto se muestran todos juntos, bajo la misma "conversación" como si de un chat se tratara.

- Agregación automática de contactos. Todos los contactos a los que se envían o reciben correos desde *Gmail*, se quedan grabados automáticamente en la agenda, sin necesidad de agregarlos uno a uno.

- Corrector ortográfico en varios idiomas (castellano, inglés,…).

- Conversaciones sin necesidad de instalar ningún software, mediante Google Talk.

- Almacenamiento de borradores de correo mientras se escribe el mismo.

- Posibilidad de contar con correo POP, que a fin de cuentas multiplica el almacenamiento al permitir descargar los mensajes al disco duro del ordenador a través de un programa gestor de correo.

- Sin presencia de publicidad, como los banners.

- Actualización automática de la bandeja de entrada cada cierto tiempo mientras se tiene abierta, sin necesidad de recargarla para mostrar los nuevos correos.

- Compatibilidad con todos los exploradores *web* y distribuciones de Sistemas Operativos.

- "*Web* clips". Aparición en la cabecera de *Gmail* de alguna de las últimas entradas de los blogs preferidos, mostrada al azar, pudiendo así estar informado de una manera sencilla de las actualizaciones de estos sitios.

- Configuración de respuesta por defecto, en largas ausencias de visita al correo.

- Personalizable. Gracias a extensiones de Firefox como *Gmail* Skins, se puede personalizar el correo al gusto de cada usuario.

- No crea favoritismos. No se ofrecen opciones extra si pagas una cuota.

- Vista del título del mensaje y primera línea. Útil para discernir entre mensajes prioritarios.

- Otras funcionalidades: PGtGM (convertidor de *Gmail* a POP3), Gtray (notificador de nuevo correo), GML (para importar mensajes) o G-Mailto, para acceder directamente a un mensaje nuevo de *Gmail* al pinchar una dirección de correo electrónico en cualquier página *web*.

2.2 Protocolos que intervienen en una aplicación de correo electrónico.

El primer protocolo que se desarrolló para la entrega de mensajes en Internet fue el protocolo SMTP (*Simple Mail Transfer Protocol*). Fue desarrollado pensando en que los sistemas que intercambiarían mensajes eran grandes computadores, de tiempo compartido y multiusuario que estaban continuamente conectados a la red Internet. Sin embargo con la aparición de los PCs en el mundo de Internet, que tenían una conectividad ocasional, se hizo necesaria una solución para que el correo llegase a estos PCs.

Para solventar esta limitación, en 1984 surgió el POP (*Post Office Protocol*). Este protocolo, en su especificación inicial, solo permitía funciones básicas como recuperar todos los mensajes, mantenerlos en el servidor y borrarlos. En sucesivas versiones del protocolo (POP2 y la actual versión POP3) se han ampliado las funciones que permiten una mejor gestión del correo.

Estos dos protocolos son los encargados de transportar el correo por toda la red Internet, pero solo son capaces de transportar mensajes en formato texto ASCII. Para superar esta limitación se utilizaban hasta hace poco tiempo, programas como el *UUEncode* y el *UUDecode*.

En 1992 surgen las MIME (*Multipurpose Internet Mail Extensions*), que permiten el correo electrónico en otras lenguas además del inglés, además de sonido, gráficos, vídeo, etc. En la actualidad el estándar MIME es el que se usa.

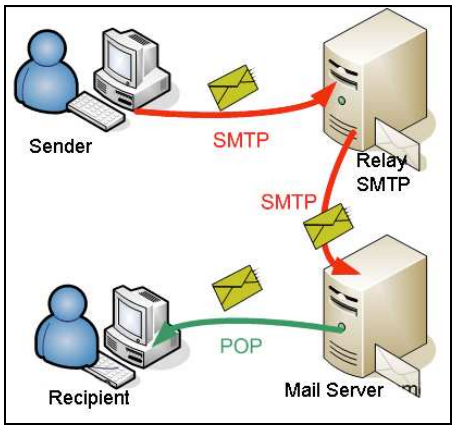


Figura 2: Envío y Recepción de un correo electrónico.

El correo electrónico es considerado el servicio más utilizado de Internet. Por lo tanto, la serie de protocolos TCP/IP ofrece una gama de protocolos que permiten una fácil administración del enrutamiento del correo electrónico a través de la red. Ambos protocolos se enmarcan dentro del nivel de aplicación en una arquitectura TCP/IP.

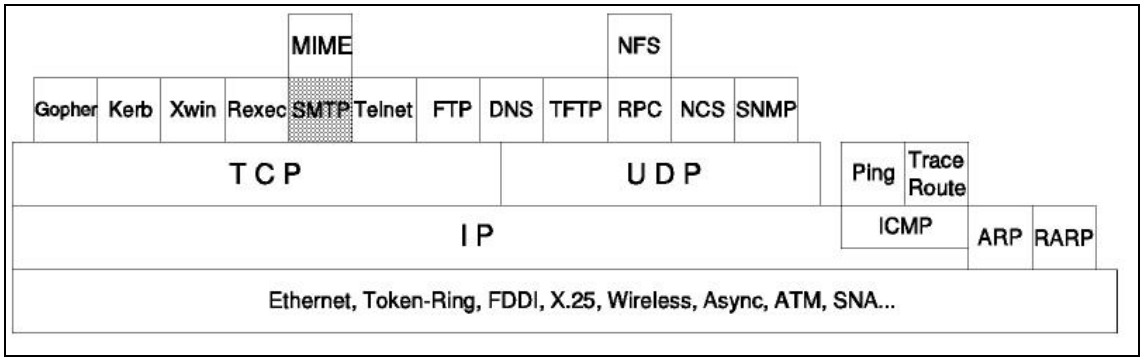


Figura 3: Capa de Comunicaciones TCP/IP.

Antes de pasar a explicar en profundidad todos los protocolos, se va a presentar un pequeño ejemplo que puede servir para una mejor comprensión de lo que se va describir a continuación:

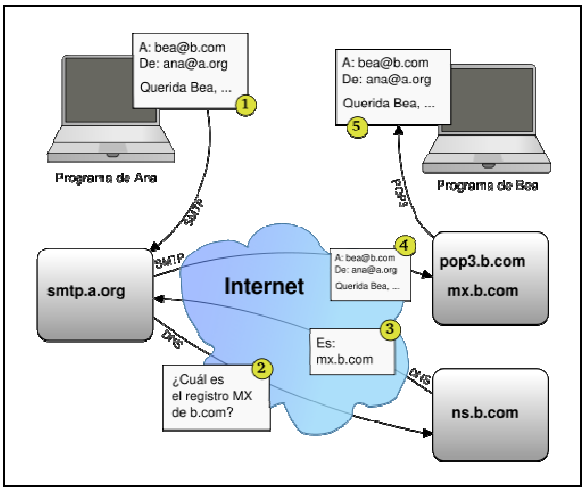


Figura 4: Ejemplo Ficticio de envío y recepción de emails.

En este ejemplo ficticio, *Ana* (**ana@a.org**) envía un correo a *Bea* (**bea@b.com**). Cada persona está en un servidor distinto (una en `a.org`, otra en `b.com`), pero éstos se pondrán en contacto para transferir el mensaje. Por pasos:

a) *Ana* escribe el correo en su programa cliente de correo electrónico. Al darle a *Enviar*, el programa contacta con el servidor de correo usado por *Ana* (en este caso, `smtp.a.org`). Se comunica usando un lenguaje conocido como protocolo SMTP. Le transfiere el correo y le da la orden de enviarlo.

b) El servidor SMTP ve que ha de entregar un correo a alguien del dominio `b.com`, pero no sabe con qué ordenador tiene que contactar. Por eso consulta a su servidor DNS (usando el protocolo DNS) y le pregunta que quién es el encargado de gestionar el correo del dominio `b.com`. Técnicamente, le está preguntando el registro MX asociado a ese dominio.

c) Como respuesta a esta petición, el servidor DNS contesta con el nombre de dominio del servidor de correo de *Bea*. En este caso es `mx.b.com`, es un ordenador gestionado por el proveedor de Internet de *Bea*.

d) El servidor SMTP (`smtp.a.org`) ya puede contactar con `mx.b.com` y transferirle el mensaje, que quedará guardado en este ordenador. Se usa otra vez el protocolo SMTP.

e) Más adelante (quizás días después), *Bea* aprieta el botón "*Recibir nuevo correo*" en su programa cliente de correo. Esto empieza una conexión, mediante el protocolo POP3 o IMAP, al ordenador que está guardando los correos nuevos que le han llegado. Este ordenador (`pop3.b.com`) es el mismo que el del paso anterior (`mx.b.com`), ya que se encarga tanto de recibir correos del exterior como de entregárselos a sus usuarios. En el esquema, *Bea* recibe el mensaje de *Ana* mediante el protocolo POP3.

Ésta es la secuencia básica, pero pueden darse varios casos especiales:

- Si ambas personas están en la misma red (una Intranet de una empresa, por ejemplo), entonces no se pasa por Internet. También es posible que el servidor de correo de *Ana* y el de *Bea* sean el mismo ordenador.

- *Ana* podría tener instalado un servidor SMTP en su ordenador, de forma que el paso 1 se haría en su mismo ordenador. De la misma forma, *Bea* podría tener su servidor de correo en el propio ordenador.

- Una persona puede no usar un programa de correo electrónico, sino un *web*mail. El proceso es casi el mismo, pero se usan conexiones HTTP al *web*mail de cada usuario en vez de usar SMTP o IMAP/POP3.

- Normalmente existe más de un servidor de correo (MX) disponible, para que aunque uno falle, se siga pudiendo recibir correo.

2.2.1 SMTP (Simple Mail Transfer Protocol).

El significado de las siglas de SMTP es Protocolo Simple de Transmisión de Correo (*Simple Mail Transfer Protocol*) [7].

Este protocolo esta descrito en la RFC 2821 y es el estándar de Internet para el intercambio de correo electrónico.

SMTP es un protocolo independiente del subsistema de transmisión usado.

Necesita que el subsistema de transmisión ponga a su disposición un canal de transmisión fiable y con entrega ordenada, con lo cual el uso del protocolo TCP en la capa de transporte es lo adecuado.

Para que dos sistemas intercambien correo mediante el protocolo SMTP, no es necesario que exista una conexión interactiva, ya que este protocolo usa métodos de almacenamiento y reenvío de mensajes.

2.2.1.1 Modelo de comunicaciones SMTP.

Cuando un sistema servidor de SMTP desea enviar uno o varios mensajes a otro servidor SMTP, el sistema emisor establece una conexión con el sistema receptor. Esta conexión es unidireccional, es decir, que el sistema emisor puede enviar correo al receptor, pero durante esa conexión el sistema receptor no puede enviar correo al sistema emisor. Si el sistema receptor tiene que enviar correo al sistema emisor, tiene que esperar a que finalice la conexión establecida y establecer otra en sentido contrario cambiando los papeles de emisor y receptor.

Una vez establecida la conexión, el sistema emisor envía comandos y mensajes. Los mensajes pueden tener como destino el sistema receptor o solo utilizarlo de intermediario para llegar a un destino más lejano. El sistema receptor puede enviar al sistema emisor respuestas y códigos de estado. Los comandos son cadenas de caracteres que se pueden entender fácilmente y las respuestas son códigos numéricos seguidos de una explicación del código, que también son legibles.

Si el sistema receptor ha recibido mensajes cuyo destino es otro, entonces establece una nueva conexión con un tercer servidor SMTP y así sucesivamente hasta que los mensajes lleguen a su destino. Si el sistema receptor tiene mensajes destinados a él, entonces los distribuye entre los distintos buzones de su sistema.

Este modelo de comunicación SMTP se representa con el siguiente gráfico:

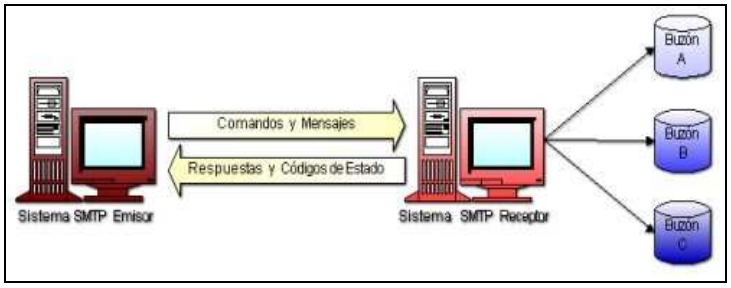


Figura 5: Modelo de comunicaciones SMTP.

2.2.1.1.1 Formato de los mensajes de Internet.

El formato de mensajes de Internet se describe en el RFC 822 "*Standard for the format of ARPA Internet Text Messages*". Aquí solo se describen las funcionalidades más interesantes para este trabajo. Para una profundización en el tema, ver el RFC anterior.

Básicamente el formato consta de un encabezado y un núcleo del mensaje. La parte dedicada al encabezado se encuentra en la parte superior de cada mensaje. Hay una línea en blanco que separa al encabezado del núcleo del mensaje. El encabezado de un mensaje consta de varias líneas, cada una de las cuales contiene un campo del encabezado. Su formato es el siguiente:

nombre_de_campo: cuerpo_de_campo#13#10

Cada campo consta como podemos ver de un nombre de campo, seguido de dos puntos (:), a continuación un espacio en blanco y el cuerpo del campo. Como final de la línea dos caracteres ASCII, **#13#10** o **CRLF** (retorno de carro y avance de línea).

Los nombres de campo no deben cambiarse ya que están estandarizados y de ellos depende la gestión de las aplicaciones de correo, tanto servidores como clientes. Además de para rutar los mensajes, el encabezado de un mensaje informa de como se debe visualizar o interpretar el contenido de un mensaje. Algunos de los campos del encabezado del mensaje son obligatorios, otros son opcionales e incluso el usuario puede definir campos no estandarizados para su uso.

El encabezado es la parte del mensaje que contiene la información necesaria para que el mensaje pueda ser transmitido y entregado desde un usuario emisor hasta un usuario receptor. El núcleo del mensaje es la parte que contiene realmente la información o mensaje que se desea enviar. Un mensaje de correo electrónico siempre se finaliza con un punto en una única línea al final del núcleo del mensaje. El formato general del mensaje es el siguiente:

nombre_de_campo1: cuerpo_de_campo1#13#10
nombre_de_campo2: cuerpo_de_campo2#13#10
...
primera linea de texto del mensaje
...
ultima linea de texto del mensaje

El Encabezado de un mensaje: Campos

Para la explicación de los campos del encabezado, se van a dividir en secciones según su funcionalidad:

Campos de Remitente

Los campos de remitente hacen referencia al origen del mensaje. Estos campos de remitente son **From**, **Sender** y **Reply-to**.

Campos de Destinatario

Los campos de destinatario indican a quien se le debe entregar el mensaje. Estos campos son **To**, **CC** y **BCC.**

Campos de Referencia

Los campos de referencia identifican al mensaje. También identifican a cualquier otro mensaje al que haga referencia. Los campos de este tipo son **Message-ID**, **In-Reply-to**, **References** y **Keywords**.

Message-ID: este campo del encabezado contiene un identificador único que es generado por la maquina remitente (el servidor). Este campo es usado por la aplicación encargada de enviar y recibir el mensaje. Este identificador es normalmente una combinación de la fecha, la hora en que se generó el mensaje,el nombre de la maquina y posiblemente un generador de números para garantizar su exclusividad.

Ej) **Message-ID**: 199707311220.0AA17101@*Gmail*.com

In-Reply-To: este campo contiene todos los ID (identificadores) de mensaje de los mensajes anteriores a los que este mensaje es respuesta. Puede contener varios ID de mensaje separados por comas y el principio y final de los ID de mensaje están delimitados por los símbolos < y >. Este campo es añadido por la aplicación cliente.

Ej) **In-Reply-To**: 199707311220.0AA17101@*Gmail*.com

References: este campo contiene todos los ID de mensaje de los mensajes anteriores a los que este mensaje hace referencia. Es un campo generado por la aplicación cliente. Puede contener uno o varios ID de mensaje separados por comas.

Ej) **References**: <199707311220.0AA17101@*Gmail*.com>

Keywords: este campo contiene palabras clase, incluso frases que identifican el contenido del mensaje. Puede ser una palabra o frase o varias separadas por comas. Este campo es introducido por el usuario.

Ej) **Keywords**: proyecto, es de correo electrónico, trabajar duro.

Campos de Seguimiento

Estos campos indican la ruta que han tenido que recorrer los mensajes para llegar a su destino. Todos los campos de este tipo son añadidos por los servidores de correo, nunca por las aplicaciones cliente. Estos campos son **Return-Path** y **Received**.

Return-Path: este campo es añadido por el último sistema que entrega el mensaje al usuario final. Contiene la ruta de acceso que ha tomado el mensaje desde el sistema origen al sistema destino.

Ej) **Return-Path**: juan@*Hotmail*.es

Received: este campo es añadido por todos los sistemas por los que pasa el mensaje antes de llegar a su destino. Normalmente esta información contenida en el campo Received se utiliza para hacer un seguimiento de los posibles problemas de transporte.
Ej) **Received**: (from pepe@localhost) by pfcemail@*Gmail*.com (8.6.12/8.6.9) id 0AA17101 for y1426936; Thu, 31 Jul 2007 14:20:58 +0200

Otros Campos

Los campos incluidos en esta sección no tienen nada que ver unos con otros como en las secciones anteriores. El objetivo de cada uno de estos campos es individual y no tienen que ver entre ellos. Estos campos son **Date**, **Subject**, **Comments** y **Encrypted**.

Date: este campo contiene la fecha y hora en la que se creó el mensaje. Este campo es añadido bien por la aplicación cliente o por el servidor de correo. El formato de la fecha en los mensajes de Internet es el siguiente:

Date: [dia,] fecha hora

El día es opcional como podemos ver, pero siempre se suele poner. El día se representa con tres caracteres del día de la semana. La fecha sigue el formato **DD MMM AAAA**. El mes se representa con tres caracteres que indican el mes. La hora tiene el formato **HH:MM[:SS] zona horaria**. HH es la hora, MM son los minutos y SS son los segundos en los que se creo el mensaje. La zona horaria se representa con tres caracteres que indican la zona horaria donde se origino la fecha y la hora.

Ej) **Date**: Mon, 28 Jul 2007 18:28:14 GMT

Subject: este campo contiene un resumen de lo que dice el mensaje. Este campo es rellenado normalmente por el remitente del mensaje (persona, proceso o sistema).

Ej) **Subject**: Como va el proyecto

Comments: este campo permite a los usuarios añadir comentarios al mensaje sin alterar el contenido de cuerpo o núcleo del mensaje.

Ej) **Comments**: esto es un comentario sin importancia

Encrypted: este campo es introducido por el usuario y le indica, si está presente, que el cuerpo o núcleo del mensaje están encriptado. El contenido de este campo indica en primer lugar el software con el que se encriptó el mensaje y en segundo lugar (es un indicador opcional) una ayuda al usuario para desencriptar el mensaje, como por ejemplo la clave publica de un sistema de encriptación de clave publica/clave privada. El encabezado de un mensaje no puede estar encriptado ya que es necesario que este legible para los servidores de correo.

Ej) **Encrypted**: PGP 123456

Campos de Extensión

Estos campos son extensiones del formato de mensajes de Internet. Estas extensiones son fundamentalmente para dar soporte a Multipurpose Internet Multimedia Extension (MIME). Estos campos son **MIME-Version**, **Content-Transfer-Encoding** y **Content-Type**.

MIME-Version: este campo indica la versión del protocolo que se está utilizando en la aplicación cliente. Por tanto es añadido por ésta.

Ej) **MIME-Version**: 1.0

Content-Transfer-Encoding: este campo es un modificador del tipo de medio del mensaje. Indica que tipo de codificación se utilizó en el mensaje y, por tanto, que tipo de descodificación se debe emplear para dejarlo en su estado normal. Es introducido por la aplicación cliente.

Ej) **Content-Transfer-Encoding**: 7bit

Content-Type: este campo indica el tipo de medio del mensaje, como puede ser text/plain o multipart/mixed. Es añadido también por la aplicación cliente.

Ej) **Content-Type**: text/plain; charset=US-ASCII

Campos definidos por el usuario

El formato de los mensajes en Internet permite que los usuarios definan sus propios campos. Estos campos definidos por el usuario siempre empiezan pro **X-** seguidos de un nombre de campo.

Ej) **X-Proyecto**: gestor de correo electronico v1.0

2.2.1.2 Comandos del protocolo SMTP

Una vez asentadas las bases del formato de los mensajes de Internet, se va a proceder a concretar el intercambio de mensajes entre un cliente SMTP y un servidor SMTP. Este diálogo se basa en un conjunto de comandos enviados por el cliente SMTP, que son palabras en formato texto ASCII legibles con facilidad y unos códigos de respuesta numéricos seguidos de un texto que explica dicho código, que son enviados por el servidor SMTP.

2.2.1.2.1 Códigos de respuesta.

Los códigos de respuesta están formados por tres dígitos, cada uno de los cuales tiene un significado.

El primero de los dígitos de un código de respuesta indica si el comando funcionó correctamente o si fallo. La siguiente tabla representa los significados del primer dígito de un código de respuesta:

Primer dígito	Significado
1	*Respuesta positiva preliminar*. Se produjo una aceptación del comando, pero se está esperando que el cliente envíe comandos de confirmación.
2	*Respuesta de finalización positiva*. El comando finalizó correctamente.
3	*Respuesta positiva intermedia*. Se aceptó el comando pero se espera a que el cliente envíe más información.
4	*Respuesta de finalización negativa temporal*. El comando ha sido rechazado, pero el cliente debería intentarlo de nuevo.
5	*Respuesta de finalización negativa permanente*. Se rechazó el comando.

Tabla 2: Código de respuesta SMTP (1er dígito)

El segundo de los dígitos de un código de respuesta, especifica la categoría de la respuesta enviada. La siguiente tabla indica el significado del segundo dígito:

Segundo dígito	Significado
0	*Sintaxis*
1	*Información*
2	*Conexiones*
5	*Sistema de correo*

Tabla 3: Código de respuesta SMTP (2º dígito)

El tercer dígito de un código de respuesta, especifica más el significado de las categorías de las respuestas.

2.2.1.2.2 Comandos SMTP.

Los comandos SMTP son variados y bastante numerosos. Antes de ver en detalle cada uno de ellos veremos como se realiza la conexión del cliente SMTP al servidor SMTP.

Para establecer una conexión, el cliente se conecta al servidor mediante el puerto TCP **25**. Cuando se establece la conexión, el cliente recibe un código de respuesta que le indica si el servidor está en disposición de aceptar la conexión y si es capaz de abrir una

sesión o bien si el servicio de correo no esta disponible en ese momento. La tabla siguiente indica los códigos de respuesta posibles:

Código	Significado
220	El servicio de correo está disponible
421	El servicio de correo no está disponible

Tabla 4: Código de respuesta a establecimiento de conexión SMTP.

A continuación pasamos a nombrar y describir brevemente los comandos SMTP. Recordar que para una información más completa y detallada, se puede consultar el RFC 2821:

HELO: este comando es el encargado de iniciar el dialogo SMTP. Este comando tiene como parámetro el nombre del cliente para establecer su identidad. El servidor responderá con un código de respuesta 250 seguido del nombre del servidor. La sintaxis de este comando es la siguiente:

HELO nombre_cliente#13#10

Los posibles códigos de respuesta a este comando están en la siguiente tabla:

Código	Significado
250	La acción solicitada se ha completado
421	El servicio no está disponible
500	Error en la sintaxis, no se pudo reconocer el comando
501	Error en la sintaxis de los parámetros del comando
504	El parámetro del comando no está implementado

Tabla 5: Código de respuesta SMTP (Comando Helo)

MAIL: este comando indica al servidor el inicio de un mensaje de correo y le indica además quien es el remitente del mensaje. La sintaxis de este comando es:

MAIL FROM: nombre_remitente@host_remitente#13#10

Los posibles códigos de respuesta a este comando están en la siguiente tabla:

Código	Significado
250	La acción solicitada se ha completado
421	El servicio no está disponible
451	Se abandonó la acción por un error de procesamiento local
452	No se produjo la acción porque el disco no tiene espacio de almacenamiento suficiente
500	Error en la sintaxis, no se pudo reconocer el comando
501	Error en la sintaxis de los parámetros del comando
552	Abandono de la acción porque se superó la reserva de espacio

Tabla 6: Código de respuesta SMTP (Comando Mail)

RCPT (destinatario): este comando indica al servidor quien es el destinatario del mensaje que se está enviando. Si el mensaje debe ir a varios destinatarios, se pueden expresar separados por comas. La sintaxis del comando es:

RCPT TO: nombre_destinatario@host_destinatario
[,nombre_destinatario@host_destinatario, ...]#13#10

Los posibles códigos de respuesta a este comando están en la siguiente tabla:

Código	Significado
250	La acción solicitada se ha completado
251	El usuario no es local, entonces se remite el mensaje a nombre-servidor
421	El servicio no está disponible
450	No se realizó la acción porque el buzón no está disponible
451	Se abandonó la acción por un error de procesamiento local
452	No se produjo la acción porque el disco no tiene espacio de almacenamiento suficiente
500	Error en la sintaxis, no se pudo reconocer el comando
501	Error en la sintaxis de los parámetros del comando
503	Secuencia de comandos incorrecta
550	La acción no se realizó porque no se ha encontrado el buzón
551	El usuario no es local, el cliente debería conectarse a nombre-servidor
552	Abandono de la acción porque se supero la reserva de espacio
553	No se realizó la operación porque la sintaxis del nombre del buzón es incorrecta

Tabla 7: Código de respuesta SMTP (Comando RCPT)

DATA: este comando indica al servidor que el texto que va a continuación del comando es ya el mensaje de correo que debe de llevarse al destinatario indicado por el encabezado del mensaje. El texto del mensaje debe de estar de acuerdo con el estándar del formato de mensaje de Internet, descrito en la **RFC 822**. Este texto del mensaje debe finalizar con un punto, que tiene que ir precedido (del comando anterior) y sucedido de los caracteres de retorno carro/avance de línea, **#13#10**. El funcionamiento es sencillo, se envía el comando y el servidor debería responder con el código de respuesta **354**. El paso siguiente es enviar el mensaje, al término del cual se debería de recibir el código de respuesta **250**. La sintaxis del comando **DATA** es la siguiente:

DATA#13#10

La tabla siguiente indica los posibles códigos de respuesta que puede recibir este comando:

Código	Significado
250	La acción solicitada se ha completado
354	Comenzar la introducción del correo, acabando con <CRLF>.<CRLF>
421	El servicio no está disponible
451	Se abandonó la acción por un error de procesamiento local
452	No se produjo la acción porque el disco no tiene espacio de almacenamiento suficiente
500	Error en la sintaxis, no se pudo reconocer el comando
501	Error en la sintaxis de los parámetros del comando
503	Secuencia de comandos incorrecta
552	Abandono de la acción porque se supero la reserva de espacio
554	Se produjo un fallo en la transacción

Tabla 8: Código de respuesta SMTP (Comando DATA)

SEND: este comando se utiliza para enviar correo a la pantalla del terminal de la sesión actual del destinatario del mensaje. No se envía el mensaje al buzón del destinatario. Este comando se utiliza fundamentalmente cuando se necesita enviar un mensaje crítico, por ejemplo, al administrador del sistema. Si el destinatario no puede recibir el mensaje, bien porque no esta en sesión, bien porque el terminal no acepta mensajes, etc. el servidor devolverá un código de respuesta al comando RCPT que debería seguir al comando SEND. La sintaxis de este comando es:

SEND FROM: nombre_remitente@host_remitente#13#10

Los posibles códigos de respuesta a este comando están en la siguiente tabla:

Código	Significado
250	La acción solicitada se ha completado
421	El servicio no está disponible
451	Se abandonó la acción por un error de procesamiento local
452	No se produjo la acción porque el disco no tiene espacio de almacenamiento suficiente
500	Error en la sintaxis, no se pudo reconocer el comando
501	Error en la sintaxis de los parámetros del comando
502	El comando no esta implementado
552	Abandono de la acción porque se superó la reserva de espacio

Tabla 9: Código de respuesta SMTP (Comando SEND)

SOML (enviar o enviar por correo): este comando funciona como el comando SEND, con la diferencia de que si la pantalla del terminal de destinatario del mensaje no puede recibir, por el motivo que sea, el mensaje, el buzón de dicho usuario recibirá de forma automática el mensaje. La sintaxis del comando es:

SOML FROM: nombre_remitente@host_remitente#13#10

Los posibles códigos de respuesta a este comando están en la siguiente tabla:

Código	Significado
250	La acción solicitada se ha completado
421	El servicio no está disponible
451	Se abandonó la acción por un error de procesamiento local
452	No se produjo la acción porque el disco no tiene espacio de almacenamiento suficiente
500	Error en la sintaxis, no se pudo reconocer el comando
501	Error en la sintaxis de los parámetros del comando
502	El comando no está implementado
552	Abandono de la acción porque se superó la reserva de espacio

Tabla 10: Código de respuesta SMTP (Comando SOML)

SAML (enviar y enviar por correo): este comando funciona igual que el comando SOML, con la diferencia de que siempre se envía el mensaje al buzón independientemente de que llegue a la pantalla del terminal o no. La sintaxis de este comando es:

SAML FROM: nombre_remitente@host_remitente#13#10

Los posibles códigos de respuesta a este comando están en la siguiente tabla:

Código	Significado
250	La acción solicitada se ha completado
421	El servicio no está disponible
451	Se abandonó la acción por un error de procesamiento local
452	No se produjo la acción porque el disco no tiene espacio de almacenamiento suficiente
500	Error en la sintaxis, no se pudo reconocer el comando
501	Error en la sintaxis de los parámetros del comando
502	El comando no está implementado
552	Abandono de la acción porque se superó la reserva de espacio

Tabla 11: Código de respuesta SMTP (Comando SAML)

EXPN (expandir): este comando se utiliza para verificar las listas de correo. Si se le pasa como parámetro un nombre de lista de correo, el servidor nos devuelve los nombres de usuario y las direcciones de los destinatarios de la lista de correo. La sintaxis de este comando es:

EXPT nombre_lista_de_correo#13#10

Los posibles códigos de respuesta a este comando están en la siguiente tabla:

Código	Significado
250	La acción solicitada se ha completado
421	El servicio no está disponible
500	Error en la sintaxis, no se pudo reconocer el comando
501	Error en la sintaxis de los parámetros del comando
502	El comando no está implementado
504	El parámetro del comando no está implementado
550	La acción no se realizó porque no se ha encontrado el buzón

Tabla 12: Código de respuesta SMTP (Comando EXPN)

HELP: este comando hace que el servidor envíe información de ayuda sobre todos los comando o sobre un comando en concreto. Su sintaxis es:

HELP [cadena-comandos]#13#10

Los posibles códigos de respuesta a este comando están en la siguiente tabla:

Código	Significado
211	El sistema tiene disponible la ayuda
214	Mensaje de información de ayuda
421	El servicio no está disponible
500	Error en la sintaxis, no se pudo reconocer el comando
501	Error en la sintaxis de los parámetros del comando
502	El comando no está implementado
504	El parámetro del comando no está implementado

Tabla 13: Código de respuesta SMTP (Comando HELP)

NOOP (no operación): este comando provoca que el servidor responda con un OK. No afecta a ningún comando enviado anteriormente o posteriormente. Se suele usar para asegurarse de que la conexión permanece activa. La sintaxis del comando es:

NOOP#13#10

Los posibles códigos de respuesta a este comando están en la siguiente tabla:

Código	Significado
250	La acción solicitada se ha completado
421	El servicio no está disponible
500	Error en la sintaxis, no se pudo reconocer el comando

Tabla 14: Código de respuesta SMTP (Comando NOOP)

RSET (reinicio): este comando le indica al servidor que abandone la transacción de correo actual, que descarte los datos introducidos del remitente, destinatario o el mensaje. RSET provoca que se vacíen y reinicien todos los *buffers* y tablas de estado. Su sintaxis es:

RSET#13#10

Los posibles códigos de respuesta a este comando están en la siguiente tabla:

Código	Significado
250	La acción solicitada se ha completado
421	El servicio no está disponible
500	Error en la sintaxis, no se pudo reconocer el comando
501	Error en la sintaxis de los parámetros del comando
504	El parámetro del comando no está implementado

Tabla 15: Código de respuesta SMTP (Comando RSET)

TURN: este comando invierte los papeles del servidor y del cliente. El cliente toma el papel de destinatario y el servidor toma el papel de remitente. Este comando se usa para recibir los mensajes de correo que desde el servidor se quiera enviar sin tener que esperar a que el servidor inicie una sesión SMTP con el cliente después de terminar la actual. Su sintaxis es:

TURN#13#10

Los posibles códigos de respuesta a este comando están en la siguiente tabla:

Código	Significado
250	La acción solicitada se ha completado
500	Error en la sintaxis, no se pudo reconocer el comando
502	El comando no está implementado
503	Secuencia de comandos incorrecta

Tabla 16: Código de respuesta SMTP (Comando TURN)

VRFY (verificar): este comando le indica al servidor que verifique que el destinatario especificado, sea un usuario real y valido por tanto, del sistema servidor. Este comando se utiliza antes de iniciar un nuevo mensaje de correo. La sintaxis de este comando es:

VRFY nombre_usuario#13#10

Los posibles códigos de respuesta a este comando están en la siguiente tabla:

Código	Significado
250	La acción solicitada se ha completado
251	El usuario no es local, entonces se remite el mensaje a nombre-servidor
421	El servicio no está disponible
500	Error en la sintaxis, no se pudo reconocer el comando
501	Error en la sintaxis de los parámetros del comando
502	El comando no está implementado
504	El parámetro del comando no está implementado
550	La acción no se realizó porque no se ha encontrado el buzón
551	El usuario no es local, el cliente debería conectarse a nombre-servidor
553	No se realizó la operación porque la sintaxis del nombre del buzón es incorrecta

Tabla 17: Código de respuesta SMTP (Comando VRFY)

QUIT: este comando le indica al servidor que el cliente no tiene mas operaciones que realizar y que se debería cerrar la conexión. El servidor responde OK y seguidamente, cierra la conexión con el cliente. Su sintaxis es:

QUIT#13#10

Los posibles códigos de respuesta a este comando están en la siguiente tabla:

Código	Significado
221	Se está cerrando la conexión
500	Error en la sintaxis, no se pudo reconocer el comando

Tabla 18: Código de respuesta SMTP (Comando QUIT)

Un aspecto importante es que el conjunto de comandos SMTP y el modelo de comunicación de este protocolo no disponen de un comando de entrada a un sistema de correo. Por este motivo, cualquier cliente de correo puede conectarse a cualquier servidor y enviar mensajes de correo, lo que hace que el sistema de correo de Internet sea vulnerable en cuanto a seguridad se refiere. Cualquier usuario de Internet puede enviar correo con una dirección de remite falsa o real de otra persona. Aunque sí es cierto que, para poder realizar esto, se debe tener un conocimiento de estos protocolos y además hay que tener en cuenta que siempre hay ficheros logs, es decir, de registro de operaciones, mediante las cuales se pueden hacer rastreos.

Como hemos visto, el protocolo SMTP consta de un gran número de comandos, aunque sólo un pequeño número son utilizados con regularidad. Por ello, se brinda un resumen de los comandos SMTP más importantes:

Comando	Significado
HELO	Identificación que utiliza la dirección IP o el nombre de dominio del equipo remitente.
MAIL FROM:	Identificación de la dirección del remitente.
RCPT TO:	Identificación de la dirección del destinatario.
DATA	Cuerpo del correo electrónico.
QUIT	Salida del servidor SMTP.
HELP	Lista de comandos SMTP que el servidor admite.

Tabla 19: Comandos SMTP más utilizados.

2.2.1.3 Pasos básicos para enviar un correo con SMTP.

Una vez visto el modelo de comunicación que usa el protocolo SMTP, el interfaz de los mensajes en Internet y los comandos, sólo queda por ver cuales son los pasos o acciones fundamentales para enviar correo.

Antes de pasar a comentar los pasos para la transmisión de un mensaje de correo electrónico, se muestra un diagrama de estados del protocolo SMTP, que muestra el envío y recepción de comandos en las diferentes etapas del proceso:

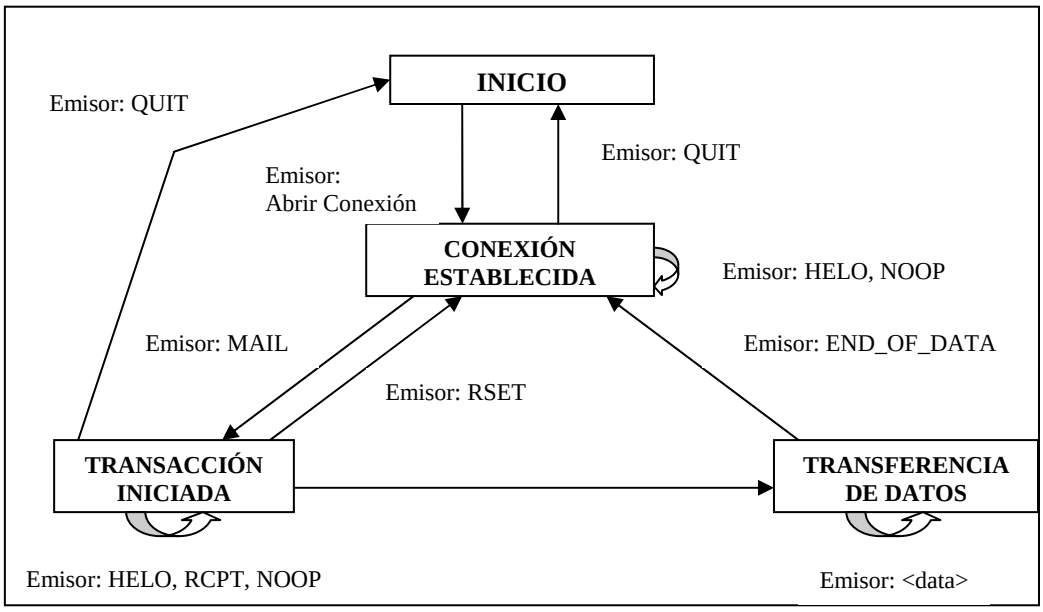


Figura 6: Diagrama de estados del protocolo SMTP.

Los pasos descritos a continuación son pasos básicos, es decir, son los pasos mínimos para enviar un mensaje de correo electrónico lo más simple posible. Los pasos son estos:

1) Conexión con el servidor de correo SMTP.

2) Recepción del saludo que envía el servidor.

3) Enviar al servidor el comando **HELO** y esperar por una respuesta.

4) Recibir la respuesta del servidor correspondiente al comando **HELO** y a continuación enviar el remitente del mensaje de correo al servidor con el comando **MAIL FROM**.

5) Recibir la respuesta del comando **MAIL FROM** y a continuación enviar el comando **RCPT TO** que permitirá indicar al servidor quien es el destinatario del mensaje.

6) Recibir la respuesta del comando **RCPT TO** procedente del servidor y se envían las copias a los destinatarios del campo **CC** con el mismo comando **RCPT TO**.

7) Una vez indicados todos los destinatarios al servidor, se envía el comando **DATA** para decirle al servidor que los datos que van a continuación son el mensaje.

8) Después de recibir la respuesta inicial al comando **DATA**, se envía al servidor el encabezado del mensaje y seguido, el núcleo o cuerpo del mensaje.

9) Se recibe la respuesta final al comando **DATA** y se desconecta del servidor a continuación con el comando **QUIT**.

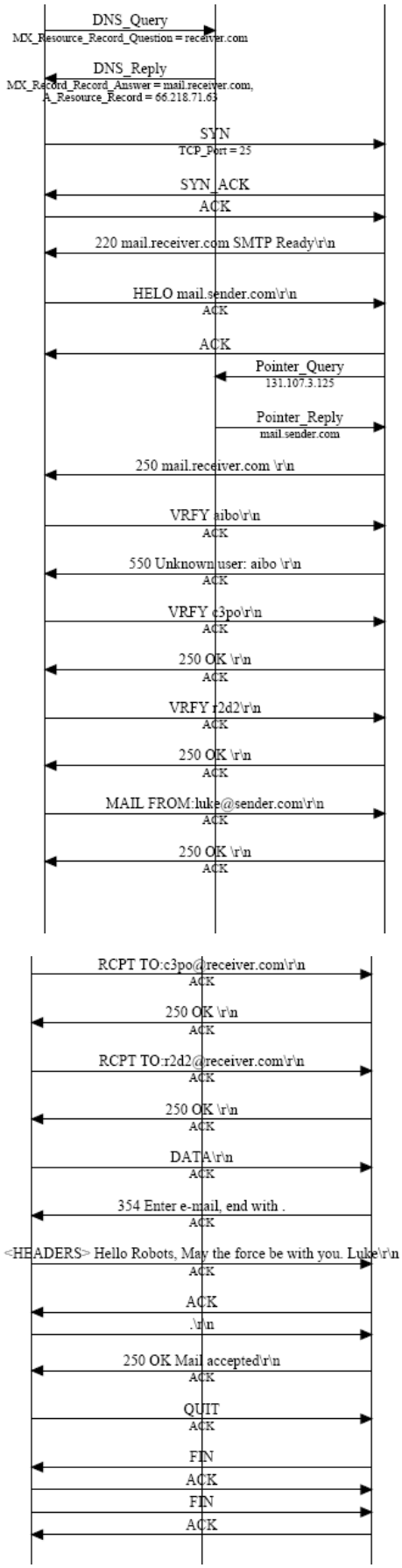
10) Cuando se recibe la respuesta del comando **QUIT** por parte del servidor, se cierra la conexión.

Por último, mostramos el extracto de una sesión Telnet donde se puede ver el envío de algunos de los comandos vistos anteriormente, así como las respuestas proporcionadas por el servidor:

```
LI:~ mc$ telnet smtp-1.dc.uba.ar 25
Trying 10.0.0.61…
Connected to smtp-1.dmz.dc.uba.ar.
Escape character is '^]'.
220 smtp-1.dc.uba.ar ESMTP
HELO milagro.dc.uba.ar
250 smtp-1.dc.uba.ar
MAIL FROM: <juan@Gmail.com>
250 ok
RCPT TO: <mcarri@dc.uba.ar>
250 ok
RCPT TO: <pepe@gil.com.ar>
250 ok
DATA
354 go ahead punk, make my day
Subject: Esto es una prueba
Hola Como estás?
Chau!
.
250 ok 1150944635 qp 24023 by smtp-1.dc.uba.ar
QUIT
221 smtp-1.dc.uba.ar Goodbye.
Connection closed by foreign host.
```

Figura 7: Ejemplo de sesión SMTP mediante un Telnet.

También se ha incluido un diagrama de mensajes de una sesión SMTP genérica entre dos servidores SMTP, *mail.sender.com* y *mail.receiver.com*. El usuario remitente de los mensajes es *Luke*, mientras que los receptores con los cuales se quiere poner en contacto son *aibo,c3po,r2d2*.



El servidor *mail.server.com* requiere un registro MX para *receiver.com.*
El servidor DNS encuentra el servidor de correo preferente para *receiver.com* y le envía a éste el nombre de dicho servidor en un registro MX que incluye también su dirección IP.

Mail.sender.com inicia la conexión TCP enviando el comando *SYN* a la dirección IP de *mail.receiver.com*, sobre el puerto 25.

La conexión TCP ha sido establecida con *mail.receiver.com.*
El receptor SMTP envía una cadena con la respuesta 220, indicando que está preparado para iniciar la transacción con el remitente.

El servidor de correo remitente se identifica mediante el commando *HELO* que contiene su nombre.

El receptor comprueba el nombre del remitente, usando la dirección IP de este último.
Respuesta que contiene el nombre del servidor de correo.

El servidor *mail.receiver.com* acepta el comando *HELO.*

Envío de la petición de verificación del cliente *aibo.*

Usuario desconocido. El correo enviado por este usuario será rechazado.
Envío de la petición de verificación del cliente *c3po.*
Usuario conocido.

Envío de la petición de verificación del cliente *r2d2.*

Usuario conocido.

Envío del comando *MAIL FROM* indicando el remitente del correo.
Confirmación por parte del receptor de la llegada del comando *MAIL FROM.*

El remitente envía el comando *RCPT TO* al receptor del correo, *c3po.*
El receptor confirma la llegada del comando.

El remitente envía el comando *RCPT TO* al receptor del correo, *r2d2.*
El receptor confirma la llegada del comando.
El remitente pide la transferencia de los datos, mediante el envío del comando *DATA.*
El receptor envía el inicio del mensaje.

Envío de las cabeceras y el cuerpo del mensaje por parte del cliente.

El punto indica el final del mensaje.

El receptor indica que el mensaje ha sido aceptado.

Envío del comando *QUIT* indicando el final de la sesión.

Fin de la conexión.

Figura 8: Diagrama de mensajes de una conexión SMTP

2.2.2 POP (Post Office Protocol).

El significado de las siglas POP es Protocolo de Oficina de Correos (*Post Office Protocol*) [8]. El protocolo visto con anterioridad, el SMTP, se creó en un momento en el que la red Internet no estaba todavía en auge. El protocolo SMTP se creo cuando los usuarios tenían cuentas en ordenadores que estaban continuamente conectados a Internet, de tal forma que cuando un usuario quería leer su correo, entraba en una sesión de terminal y solicitaba al servidor que le diese el correo que tenía almacenado para él. Esta situación ha cambiado considerablemente hoy en día. Los usuarios se conectan a la maquina servidora de correo por un periodo de tiempo muy breve, el suficiente para solicitar el envío del correo mediante un programa cliente. Por tanto el servidor de correo electrónico debe mantener almacenado el correo en sus buzones y enviarlo a los clientes de correo cuando estos se conecten y lo soliciten. Este es el objetivo para el cual se creo el protocolo Post Office Protocol, POP.

La situación actual es que se utiliza el protocolo SMTP para el envío de correo y para la recepción de correo se utiliza el protocolo POP, pero ya en su tercera versión desde su aparición, el POP3.

2.2.2.1 Modelo de comunicaciones POP.

La descripción del protocolo POP y su modelo de comunicaciones está en el documento oficial estándar RFC 1725. Este modelo de comunicaciones se basa en el concepto de buzón. Al igual que ocurre en una oficina de correos local, de una ciudad, tiene un espacio para almacenar el correo, las cartas, hasta que se recojan. De igual manera el servidor POP almacena el correo electrónico en buzones hasta que un programa cliente lo recupera.

El cliente POP se conecta con el servidor, en la mayoría de los casos, a través del puerto TCP 110. Sin embargo para acceder al servicio POP de *Gmail*, se utilizará le puerto 995. Para entrar en el servidor es necesario una cuenta de identificación en dicha máquina (lo que le permite tener un espacio reservado para su correo). A continuación es necesario verificar que se es dueño de la cuenta a través de una clave. Una vez que se ha entrado en el sistema, el cliente POP puede dialogar con el servidor para conocer si tiene correo, cuantos mensajes tiene, que se los envíe, que los borre, etc.

Para poder ofrecer estas funciones, el modelo de comunicación POP se basa en estados. Estos estados son estado de autorización, estado de transacción y estado de actualización. Después de establecer la conexión, el servidor POP se encuentra en un *estado de autentificación* o *autorización*, esperando a que el cliente le envíe el nombre y clave de la cuenta de usuario. Cuando se verifica que el nombre y la clave son correctos, el servidor pasa a un *estado de transacción*. Antes de pasar a este estado, el servidor POP hace un bloqueo del buzón para impedir que los usuarios modifiquen o borren el correo antes de pasar al estado siguiente. En este estado el servidor atiende a las peticiones del cliente. Después de enviar al servidor el comando QUIT, que luego veremos, el servidor pasa al *estado de actualización*, en este estado el servidor elimina los mensajes que estaban con la marca de borrado y finaliza la conexión.

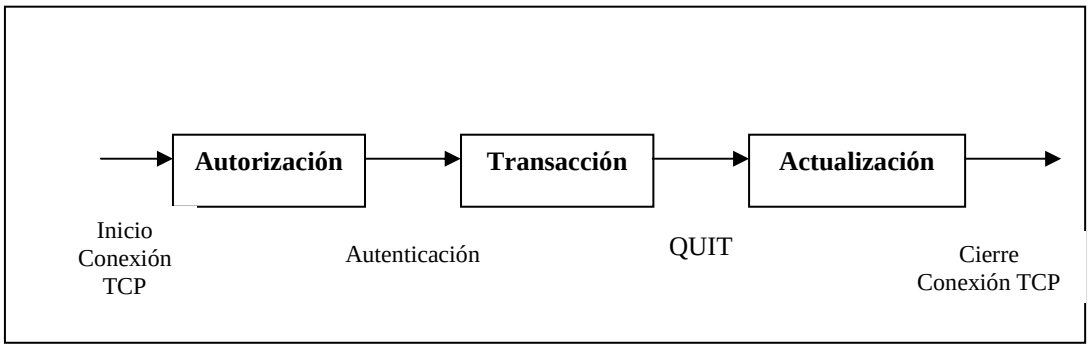


Figura 9: Diagrama de estados del protocolo POP3.

2.2.2.2 Comandos del protocolo POP.

Los diseñadores decidieron establecer un conjunto de comandos POP reducido, lo más simple posible, añadiendo en versiones posteriores del protocolo algunas funcionalidades adicionales. El diálogo entre el cliente y el servidor, se basa en el envío de comandos y el servidor responde con un código de repuesta y pasando a un estado.

2.2.2.2.1 Códigos de respuesta.

Los diseñadores del protocolo decidieron que lo importante en este protocolo era conocer si los comandos funcionaban o no, por tanto solo establecieron dos códigos de respuesta, uno para cuando el comando funciona correctamente y otro para cuando no funciona correctamente. Los códigos de respuesta que el servidor POP envía van seguidos de una frase que explica o aclara el por que de ese código, lo que nos puede ayudar a conocer cual es el motivo de los errores, si se producen.

Código	Significado
+OK	El comando funcionó correctamente
+ERR	El comando falló

Tabla 20: Códigos de respuesta POP3.

2.2.2.2.2 Comandos POP.

Los comandos POP los podemos agrupar según el estado en el que se encuentre el servidor, así tendremos comandos del estado de autorización, comando del estado de transacción, comandos del estado de actualización y comandos opcionales:

2.2.2.2.2.1 Comandos del estado de Autorización.

Al conectarse a un servidor POP, éste entra en un estado de autorización. El cliente de correo debe enviarle el nombre de la cuenta y la clave para poder continuar. Si son correctos, el buzón correspondiente a esa cuenta se pasa a un estado de bloqueo exclusivo para impedir que los mensajes sean modificados o borrados antes de llegar al estado de actualización del servidor POP. Si no se consigue pasar el buzón al estado de bloqueo exclusivo, se produce un fallo y no se puede pasar al estado de transacción.

USER: este comando le proporciona o le indica al servidor, el identificador o nombre de la cuenta de usuario. Si ese identificador existe, devuelve un código de respuesta de operación correcta, sino devuelve un código de respuesta de fallo.

USER id-cuenta#13#10

PASS (clave): este comando le indica al servidor la clave de la cuenta de usuario indicada por el comando USER. Si la clave no es correcta o el buzón no se pudo pasar al estado de bloqueo exclusivo, se produce un error. La sintaxis de este comando es la siguiente:

PASS clave#13#10

QUIT: este es un comando que se puede usar cuando el servidor está en estado de autorización y en estado de transacción. Si se usa cuando está en estado de autorización, la sesión finaliza y se interrumpe la conexión. Si se usa cuando está en estado de transacción, se cierra la sesión y pasa el servidor pasa a estado de actualización. La sintaxis de este comando es la siguiente:

QUIT#13#10

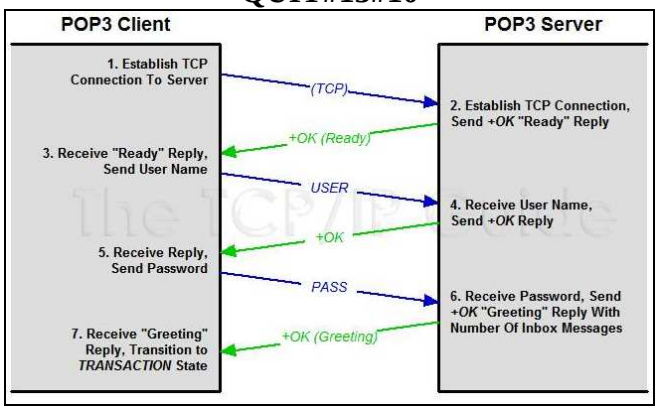


Figura 10: Secuencia de comandos del Estado de Autorización POP3.

2.2.2.2.2.2 Comandos del estado de Transacción.

En este estado, el cliente puede enviar comandos para tratar el correo, conocer si tiene o no, correo nuevo, borrar correo (marcar como borrado), recuperarlo, almacenarlo, etc.

DELE (eliminar): este comando marca como eliminado un mensaje, pero en realidad el servidor no lo elimina hasta que no pasa al estado de actualización, con lo cual no se perderían en el caso de que la conexión fallase o que quisiéramos quitarle la marca de eliminar. Cada mensaje que está en el buzón del servidor POP tiene asignado un número, que es el número con el cual identificaremos a los mensajes como en este caso para marcarlo como borrado. La sintaxis es la siguiente para este comando:

DELE numero_mensaje#13#10

LIST: este comando recupera información a cerca del tamaño que ocupa un mensaje determinado o sobre todos los mensajes. En el caso de que se aplique sobre un solo mensaje, el servidor responde con una línea, indicando el número de mensaje y el tamaño. Si no se envía con un número de mensaje, el servidor responde enviando una línea por cada mensaje con el numero y tamaño. El final de estas líneas es un punto precedido y seguido por los caracteres #13#10. La sintaxis de este comando es:

LIST [numero_mensaje]#13#10

NOOP (no operación): es un comando de no operación que cuando se envía, el servidor responde con un **OK**. Se utiliza para mantener activa la sesión. La sintaxis del comando es la siguiente:

NOOP#13#10

RETR (recuperar): este es un comando para recuperar o solicitar que el servidor envíe un mensaje determinado. El mensaje se solicita enviando el número de mensaje a continuación del comando. Este número de mensaje no puede corresponder a un mensaje con marca de borrado. El servidor responde a la petición enviando el texto del mensaje, que finaliza cuando le llega al cliente un punto seguido y precedido de los caracteres de retorno de carro/avance de línea (#13#10). La sintaxis del comando es:

RETR numero_mensaje#13#10

RSET (reiniciar): este comando anula la marca de borrado de todos los mensajes que tengan dicha marca en el buzón. No se puede eliminar la marca de borrado de un mensaje en concreto, tiene que ser de todos. La sintaxis es la siguiente:

RSET#13#10

STAT (estado): este comando obtiene un resumen del contenido del buzón. El servidor responde a este comando enviando el numero de mensajes que hay en el buzón, sin contar aquellos que están marcados como borrados, y el volumen o tamaño en bytes del buzón. Estos dos datos los devuelve separados por espacios. La sintaxis de este comando es:

STAT#13#10

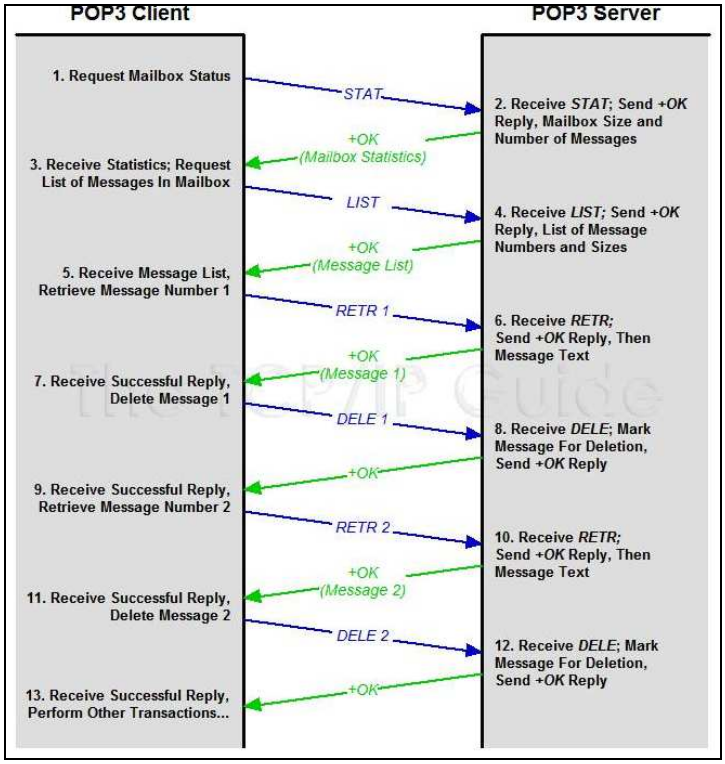


Figura 11: Secuencia de comandos del Estado de Transacción POP3.

2.2.2.2.2.3 Comandos del estado de Actualización.

En este estado no hay comandos. A este estado se llega desde es estado de transacción cuando enviamos al servidor el comando QUIT. En el estado de actualización se eliminan los mensajes que han sido marcados en el estado anterior. A continuación se le quita el bloqueo exclusivo al buzón para que pueda actualizarse dicho buzón con nuevo correo. Por ultimo, el servidor termina la conexión.

2.2.2.2.2.4 Comandos POP opcionales.

Los comandos vistos hasta ahora, son los comandos básicos necesarios, pero hay otros comandos que proporcionan una mayor flexibilidad en el cliente sin complicar en exceso el protocolo.

APOP (entrar en el sistema con contraseña encriptada): este comando es una alternativa a los comandos USER y PASS. El comando APOP necesita de dos parámetros, uno es un identificador de cuenta y el otro es la clave encriptada. Al conectarse al servidor POP, éste envía una bienvenida que incluye una marca de tiempo.

Con esta marca de tiempo se aplica el **algoritmo MD5** para encriptar la contraseña. Este algoritmo se encuentra definido en el **RFC 1321**. Este método de autentificación POP se recomienda para aquellos usuarios que se conectan frecuentemente a sus servidores de correo, evitando que la clave de la cuenta viaje frecuentemente por la red sin encriptar. La sintaxis de este comando es la siguiente:

APOP id_cuenta clave_encriptada#13#10

TOP: este comando permite al cliente de correo recuperar la parte del encabezado del mensaje y un número de líneas del cuerpo o núcleo del mensaje. Este comando se suele utilizar cuando se desea conocer los mensajes sin leerlos. La sintaxis de este comando es:

TOP numero_mensaje numero_lineas_del_cuerpo#13#10

UIDL (lista de identificadores únicos): este comando obtiene del servidor un ID de mensaje único y persistente para uno o todos los mensajes del buzón. El servidor genera un ID de mensaje que se debe de conservar entre las distintas sesiones. De esta forma, con esta identificación de mensaje única, el cliente puede realizar un seguimiento de que mensajes se han recuperado ya y cuales son nuevos. La respuesta del servidor a este comando es una línea con el número de mensaje y el identificador único en el caso de que sea para un mensaje. Para el caso de que sea para todos los mensajes, el servidor devuelve una línea por mensaje, al final de las líneas un punto precedido y seguido de los caracteres de retroceso de carro/avance de línea. La sintaxis de este comando es:

UIDL [numero_mensaje]#13#10

Al igual que en el protocolo SMTP, POP3 presenta un gran número de comandos, por ello, a continuación se muestra una tabla resumen de los comandos más utilizados:

Comando	Significado
USER identification	Permite la autenticación. Debe estar seguido del nombre de usuario, es decir, una cadena de caracteres que identifique al usuario en el servidor. El comando USER debe preceder al comando *PASS*.
PASS password	Permite especificar la contraseña del usuario cuyo nombre ha sido especificado por un comando *USER* previo.
STAT	Información acerca de los mensajes del servidor.
RETR	Número del mensaje que se va a recoger.
DELE	Número del mensaje que se va a eliminar.
LIST [msg]	Número del mensaje que se va a mostrar.
NOOP	Permite mantener la conexión abierta en caso de inactividad.
TOP <messageID> <n>	Muestra *n* líneas del mensaje, cuyo número se da en el argumento. En el caso de una respuesta positiva del servidor, éste enviará de vuelta los encabezados del mensaje, después una línea en blanco y finalmente las primeras *n* líneas del mensaje.
UIDL [msg]	Solicitud al servidor para que envíe una línea que contenga información sobre el mensaje que eventualmente se dará en el argumento. Esta línea contiene una cadena de caracteres denominada *unique identifier listing (lista de identificadores únicos)* que permite identificar de manera única el mensaje en el servidor, independientemente de la sesión. El argumento opcional es un número relacionado con un mensaje existente en el servidor POP, es decir, un mensaje que no se ha borrado.
QUIT	Solicita la salida del servidor POP3. Lleva a la eliminación de todos los mensajes marcados como eliminados y envía el estado de esta acción.

Tabla 21: Resumen de los comandos POP3 más importantes.

2.2.2.3 Pasos básicos para recibir un correo electrónico mediante POP.

Los pasos descritos a continuación son pasos básicos que permiten bajarse el correo pendiente de un servidor **POP**. Estos pasos son:

1) Conectar con el servidor **POP**.

2) Recibir el saludo del servidor y enviar el comando **USER** con el nombre de la cuenta de usuario necesaria.

3) Recibir la respuesta al comando **USER** y enviar el comando **PASS** con la clave de la cuenta que nos permitirá entrar.

4) Recibir la respuesta del comando **PASS** y enviar el comando **STAT** para averiguar cuantos mensajes hay disponibles en el buzón.

5) Cuando se recibe la respuesta al comando **STAT**, pueden darse varias posibilidades. Una puede ser emplear el comando **TOP** para obtener los encabezados de los mensajes y posteriormente elegir que mensaje se desea traer con el comando **RETR** o bien se pueden traer uno o varios, incluso todos los mensajes enteros con el comando **RETR**.

6) Después de recibir la respuesta del comando enviado en el paso anterior, y de recibir un mensaje, o todos, se pueden eliminar del buzón o dejarlos. Lo más normal es borrar aquellos mensajes que ya se hayan recuperado para que no ocupen espacio. Esto se hace con el comando **DELE**.

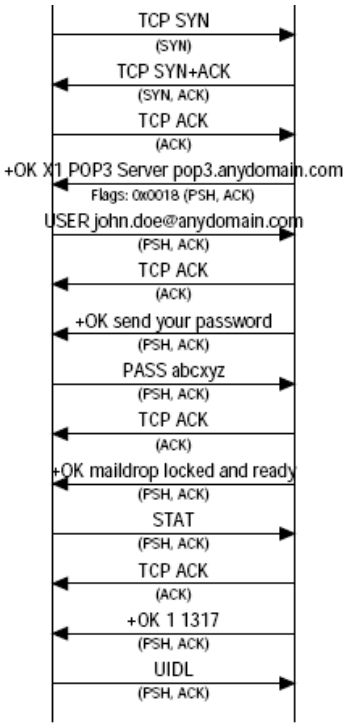
7) Por último sólo queda enviar el comando **QUIT** para finalizar la sesión y pasar el servidor al estado en el que tiene que eliminar los mensajes marcados como borrados.

8) El cliente recibe la respuesta al comando **QUIT** y finaliza la conexión.

Por último, al igual que en el caso del protocolo SMTP se muestra un ejemplo de acceso Telnet a un servidor de correo utilizando los comandos estudiados y el diagrama de mensajes de dicha transacción:

```
Server: +OK X1 POP3 Server pop3.anydomain.com
Client: USER john.doe@anydomain.com
Server: +OK send your password
Client: PASS abcxyz
Server: +OK maildrop locked and ready
Client: STAT
Server: +OK 1 1317
Client: UIDL
Server: +OK 1 messages (1317 octets)
1 390953946
.
Client: LIST
Server: +OK 1 messages (1317 octets)
1 1317
.
Client: RETR 1
Server: +OK 1317 octets
Server: E-mail contents
.
Client: DELE 1
Server: +OK msg deleted
Client: QUIT
Server: +OK POP3 Server saying Good-Bye
```

Figura 12: Ejemplo sesión POP3 mediante Telnet.



El cliente POP3 inicia el *Three Way Handshake* para establecer la conexión TCP con el servidor.

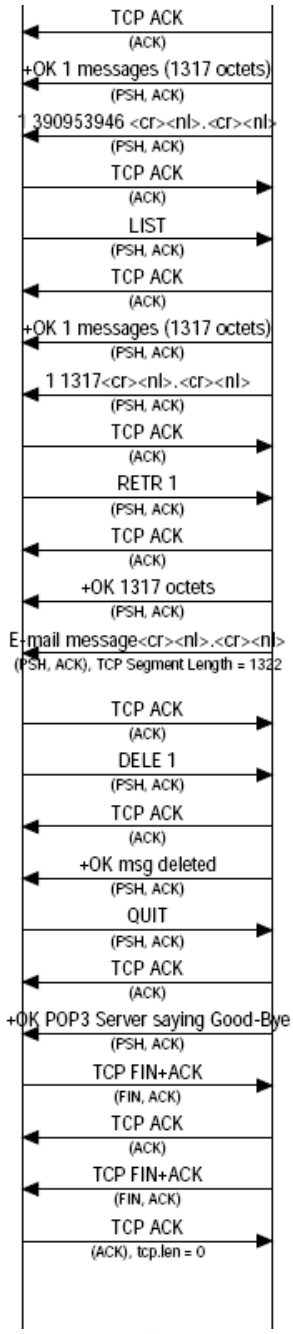
Mediante el envío de la respuesta "+OK" al cliente, el servidor reconoce el establecimiento de la conexión TCP.
El cliente POP3 envía el comando *USER* como su dirección de correo.

El servidor POP3 confirma la recepción del comando *USER* y envía la petición de la contraseña del cliente.

El cliente POP3 envía la contraseña en claro al servidor.
.
El servidor está listo para las peticiones de mensajes por parte del cliente.

El cliente envía el comando *STAT* para obtener el número de mensajes sin leer presentes en el servidor.

El servidor POP3 responde que hay un mensaje y que la capacidad total de los mensajes almacenados es de 1317 bytes.
El cliente envía el comando *UIDL* pata todos los mensajes almacenados en el servidor.



El servidor POP3 responde que tiene un correo con ID 1 cuya longitud es de 1317 bytes.
El servidor envía el ID asociados a ese mensaje.

El cliente POP3 envía el comando *LIST* al servidor.

El servidor responde al comando *LIST* con el número total de mensajes presentes y el tamaño de cada uno.
El servidor POP3 envía al cliente los mensajes presentes en el servidor así como su tamaño.

El cliente envía el comando *RETR* solicitando el mensaje 1 al servidor.

El servidor POP3 confirma la petición del cliente y le indica que dicha mensaje tiene un tamaño de 1317 bytes.
El mensaje requerido por el cliente es enviado por el servidor a través de un segmento TCP.

El mensaje 1 ha sido almacenado en el equipo del cliente, por lo que éste envía el comando *DELE* para eliminar el mensaje del servidor.

El servidor confirma la eliminación del mensaje.

El cliente envía el comando *QUIT* para iniciar el final de la sesión.

El servidor confirma la recepción del comando *QUIT*.

Cierre de la conexión entre el cliente POP3 y el servidor.

Figura 13: Diagrama de mensajes de ejemplo de sesión Telnet.

2.2.3 IMAP (Internet Message Access Protocol)

2.2.3.1 Descripción del protocolo IMAP

IMAP (**I**nternet **M**essage **A**ccess **P**rotocol) [9] es un protocolo de red de acceso a mensajes electrónicos almacenados en un servidor. Mediante IMAP se puede tener acceso al correo electrónico desde cualquier equipo que tenga una conexión a Internet. IMAP tiene varias ventajas sobre POP, que es el otro protocolo empleado para obtener correo desde un servidor, como hemos visto anteriormente. Por ejemplo, es posible especificar en IMAP carpetas del lado servidor. Por otro lado, es más complejo que POP.

La versión actual de IMAP, IMAP versión 4 revisión 1 (IMAP4ver1), está definida por el *RFC 3501*.

IMAP es utilizado frecuentemente en redes grandes; por ejemplo los sistemas de correo de un campus. IMAP permite a los usuarios acceder a los nuevos mensajes instantáneamente en sus computadoras, ya que el correo está almacenado en la red. Con POP3 los usuarios tendrían que descargar el email a sus computadoras o accederlo vía *web*. Ambos métodos toman más tiempo de lo que le tomaría a IMAP, y se tiene que descargar el email nuevo o refrescar la página para ver los nuevos mensajes.

De manera contraria a otros protocolos de Internet, IMAP4 soporta mecanismos nativos de cifrado. La transmisión de contraseñas en texto plano también es soportada.

2.2.3.2 Comandos más relevantes del protocolo IMAP.

A continuación se muestra una tabla resumen con los comandos más relevantes del protocolo IMAP, así como una pequeña explicación de su funcionamiento.

Comando	Significado
CAPABILITY	Pide la lista de funcionalidades disponible.
AUTHENTICATE	Especifica un mecanismo de autenticación.
LOGIN	Proporciona el nombre de usuario y la contraseña.
SELECT	Especifica el buzón de correo.
EXAMINE	Especifica el buzón de correo sólo en modo lectura.
CREATE	Crea un buzón de correo.
DELETE	Elimina un buzón de correo.
RENAME	Renombra un buzón de correo.
SUBSCRIBE	Añade un buzón de correo a la lista activa de buzones.
UNSUBSCRIBE	Elimina un buzón de correo de la lista activa de buzones.
LIST	Muestra los buzones de correo.
LSUB	Muestra los buzones de correo presentes en la lista activa.
STATUS	Estado de los buzones (nº de mensajes,…)
APPEND	Añade un mensaje al buzón de correo.
CHECK	Inserta un punto de chequeo en el buzón de correo.
CLOSE	Procede a la eliminación de los mensajes y cierra el buzón.
EXPUNGE	Procede a la eliminación de los mensajes.
SEARCH	Busca en los buzones el mensaje especificado por algún criterio.
FETCH	Obtiene alguna parte del mensaje especificado.
STORE	Cambia datos de un mensaje especificado.
COPY	Copia un mensaje a otro buzón.
NOOP	No hace nada relevante.

LOGOUT	Cierra la conexión.

Tabla 22: Tabla resumen de los comandos IMAP más importantes.

2.2.3.3 Ventajas sobre POP3.

Algunas de las características importantes que diferencian a IMAP y POP3 son:

- Soporte para los modos de operación *connected* y *disconnected* :
    - Al utilizar POP3, los clientes se conectan al servidor de correo brevemente, solamente el tiempo necesario para descargar los nuevos mensajes. Al utilizar IMAP, los clientes permanecen conectados el tiempo que su interfaz permanezca activa y descargan los mensajes bajo demanda. Esta forma de trabajar de IMAP puede dar tiempos de respuesta más rápidos para usuarios que tienen una gran cantidad de mensajes o mensajes grandes.

- Soporte para la conexión de múltiples clientes simultáneos a un mismo destinatario:
    - El protocolo POP3 asume que el cliente conectado es el único dueño de una cuenta de correo. En contraste, el protocolo IMAP4 permite accesos simultáneos a múltiples clientes y proporciona ciertos mecanismos a los clientes para que se detecten los cambios hechos a un mailbox por otro cliente concurrentemente conectado.

- Soporte para acceso a partes MIME de los mensajes y obtención parcial:
    - Casi todo el correo electrónico de Internet es transmitido en formato MIME. El protocolo IMAP4 le permite a los clientes obtener separadamente cualquier parte MIME individual, así como obtener porciones de las partes individuales o los mensajes completos.

- Soporte para que la información de estado del mensaje se mantenga en el servidor:
    - A través de la utilización de *banderas* definidas en el protocolo IMAP4 de los clientes, se puede vigilar el estado del mensaje, por ejemplo, si el mensaje ha sido o no leído, respondido o eliminado. Estas *banderas* se almacenan en el servidor, de manera que varios clientes conectados al mismo correo en diferente tiempo pueden detectar los cambios hechos por otros clientes.

- Soporte para accesos múltiples a los buzones de correo en el servidor:
    - Los clientes de IMAP4 pueden crear, renombrar o eliminar correo (por lo general presentado como carpetas al usuario) del servidor, y mover mensajes entre cuentas de correo. El soporte para múltiples buzones de correo también le permite al servidor proporcionar acceso a los directorios públicos y compartidos.

- Soporte para búsquedas de parte del servidor:
    - IMAP4 proporciona un mecanismo para que los clientes pidan al servidor que busque mensajes de acuerdo a una cierta variedad de criterios. Este mecanismo evita que los clientes descarguen todos los mensajes de su buzón de correo con el fin de agilizar las búsquedas.

- Soporte para un mecanismo de extensión definido.
- Como reflejo de la experiencia en versiones anteriores de los protocolos de Internet, IMAP define un mecanismo explícito mediante el cual puede ser extendido. Se han propuesto muchas extensiones de IMAP4 y son de uso común.

- Un ejemplo de extensión es el IMAP IDLE, que sirve para que el servidor avise al cliente cuando ha llegado un nuevo mensaje de correo y éstos se sincronicen. Sin esta extensión, para realizar la misma tarea, el cliente debería contactar periódicamente al servidor para ver si hay mensajes nuevos.

Por último, se muestra una tabla comparativa, de las características más relevantes de cada protocolo:

Característica	POP3	IMAP
Ubicación de los mensajes una vez descargados	En el equipo local. (Por defecto)	En el servidor de correo.
Acceso a los correos descargados	Sólo en el equipo local.	En cualquier equipo con conexión a Internet.
Capacidad Limitada	No, sólo la capacidad del disco duro del equipo del usuario.	Sí, hay que eliminar periódicamente los correos del servidor.
Velocidad	Lento. Descarga todos los mensajes al equipo local.	Rápido. Sólo se extraen las cabeceras del mensaje.
Útil para…	Usuarios que accedan desde un lugar fijo.	Usuarios que quieran acceder desde distintos lugares o equipos.
Descarga de Adjuntos	Sólo se descargan una vez. (Rápido)	Se descargan cada vez que se quieren ver (Lento).
Modo de Operación	Desconectado	Conectado.
Número de Clientes	Único	Multicliente
Criterio para búsqueda de mensajes	No.	Sí.

Tabla 23: Tabla comparativa entre los protocolos IMAP y POP3

2.2.3.4 Motivos de la elección de POP3 frente a IMAP.

Una de las principales razones por las que se ha elegido el protocolo POP3 frente a IMAP para realizar este proyecto, es que POP3 permite a descarga de correos directamente a un PC, sin la necesidad de estar permanentemente conectado al servidor de correo. Esto es interesante en este proyecto, ya que está pensado para aplicaciones de

recogida de datos, que no pueden estar continuamente conectadas al servidor de correo, bien por problemas de conectividad o limitaciones en el consumo de energía.

Partiendo de lo anterior, POP3 está pensado para la consulta de correo en un lugar determinado, otro punto a tener en cuenta en el proyecto, ya que siempre querremos procesar los correos desde el mismo lugar.

Otra de las razones, es que hasta hace muy poco, el servidor de correo de *Gmail*, no proporcionaba el servicio IMAP, por lo que este servicio no está lo suficientemente probado y todavía no es implementado por muchos usuarios.

Para finalizar se muestra una tabla comparativa de ambos protocolos para el caso de utilizar cuentas de correo de *Gmail*:

Gmail by Google BETA

	IMAP	POP
Gratis para usuarios de Gmail	X	X
Permite leer correo en clientes y telefonos	X	X
Permite usar etiquetas de Gmail en tu cliente/celular	X	
Sincroniza cambios con Gmail, nos ahorra trabajo	X	
Sincroniza cambios a través de multiples dispositivos	X	

Figura 14: Tabla comparativa de los protocolos POP3 e IMAP para el caso del proveedor de correo *Gmail.*

2.2.4 Tecnología MIME (Multipurpose Internet Mail Extensions).

2.2.4.1 Descripción de MIME.

MIME es acrónimo de *Multipurpose Internet Mail Extensions Encoding* [10][11][12], un estándar utilizado en Internet con dos finalidades: de un lado, normalizar el intercambio de todo tipo de archivos (texto, audio, vídeo, etc) en la Red; de otra, acabar con el problema de las transferencias de texto internacional por e-mail.

En esencia, el camino seguido para transmitir cualquier fichero es siempre el mismo: transformar (codificar) el fichero no ASCII en US-ASCII (haciéndolo al mismo tiempo compatible con el estándar SMTP *Simple Mail Transport Protocol*); transmitirlo en este formato, y reconvertirlo en destino al formato original (decodificarlo).

Para lograrlo, de entre la enorme variedad de tipos de formatos (codecs) de ficheros utilizados, se acordó establecer unas tablas lo más completas posibles en las que se indique que codificación debe utilizarse para cada tipo y subtipo. El sistema tiene la ventaja de que cuando aparece un nuevo tipo de formato (es frecuentísimo) solo hay que actualizar en los Navegadores y Agentes de Correo los pares de valores tipo-de-fichero/tipo-de-codec. Para evitar una excesiva proliferación de codificadores distintos, la IANA se encarga de establecer las referidas tablas.

El estándar no solo ha sido integrado en los programas de correo, sino que se ha integrado en el protocolo HTTP de la *Web*, de forma que los servidores *Web* pueden identificar los tipos de ficheros que tienen que enviar a sus clientes. A su vez los navegadores pueden conocer la codificación a aplicar con cada uno de los contenidos recibidos.

En lo que se refiere al correo electrónico, una de las características principales de los sistemas de codificación MIME es que se han diseñado de forma que se mantenga sin modificación la mayor parte posible de texto (todos los caracteres que sean US-ASCII se transmitan sin modificación).Sólo son codificados aquellos caracteres "no estándar" que puedan tener problemas en alguna parte del sistema Internet. De esta forma, si algún agente de correo no es conforme a MIME, el texto codificado todavía tendrá algún sentido (lo que ocurre cuando recibimos, o nos dicen que reciben, esos caracteres extraños intercalados en nuestros correos).

2.2.4.2 Relación entre MIME y el correo electrónico.

Como se ha señalado al tratar de los protocolos de transmisión utilizados en el correo electrónico, la porción de mensaje que vemos está completada con una cabecera en la que el sistema incluye información adicional. Reproducimos aquí el cuadro con algunas de estas "etiquetas" incluidas en cualquier mensaje.

```
    From: "A.J.Millan" <ajmillan@ctv.es>
To: <echo@rediris.es>
Subject: =?iso-8859-1
?B?RXN0YSBsZW5n/GEg8W/xYSBubyBzZSBpcuEgY29uIOlsLiA=?=
Date: Wed, 2 Sep 1998 13:00:57 +0200
MIME-Version: 1.0
Content-Type: text/plain;
charset="iso-8859-1"
Content-Transfer-Encoding: 8bit
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express 4.72.2106.4
X-MimeOLE: Produced By Microsoft MimeOLE V4.72.2106.4
```

- MIME- Version 1: indica al programa de correo que se trata de un mensaje en dicho formato (es decir, que se ajusta a lo señalado en la RFC 1521.

- Content-Type: Indica que tipo de datos contiene el mensaje. Pueden encontrarse los siguientes:

    - text/plain: solo texto.
    - Image: imágenes estáticas.
    - Video: imágenes dinámicas, puede incluir audio.
    - Audio: sonido.
    - Message: significa que el contenido está configurado según el estándar RFC 822, esto puede ser usado para reexpedir mensajes.
    - Application: se emplea para señalar que el contenido se enviará a un programa externo, por ejemplo, texto para una impresora *PostScript.*

Un mensaje también puede tener varias partes (*multipart*) con varios contenidos separados, incluso de tipos diferentes (texto, audio e imágenes). Incluso cada parte puede tener subpartes (ser a su vez multiparte), puesto que el formato MIME puede ser recursivo.

Content-type puede tener un subtipo, ambos separados por una barra inclinada "/". Por ejemplo, *image/gif* es una imagen en formato *GIF*; el tipo es *image* y el subtipo *gif*.
Finalmente, puede tener parámetros opcionales empezando por un punto y coma ";". Por ejemplo, el parámetro *charset* en *Content-type=text/plain, charset=iso-8859-1*, indica que el cuerpo del mensaje utiliza el juego de caracteres ISO-8859-1 (El juego de caracteres por defecto es US-ASCII).

Content-transfer-encoding: Señala como ha sido codificado el mensaje para su transmisión por e-mail, de forma que pueda viajar sin problemas de que sea corrompido desde el destinatario al receptor a través de los agentes de correo (**MUA**s). Los tipos que pueden darse para esta etiqueta son los anteriormente definidos (tipos de codificación posibles) es decir: *7bit, quoted-printable, base 64, 8 bit* y *binary.*

Nota: recientemente ha aparecido una variación del protocolo original denominado **S/MIME**, destinado a garantizar la seguridad de las comunicaciones y basado en el estándar de seguridad **RSA** Data Security. Aunque en este sentido compite con otra tecnología, **PGP** ("Prety Good Privacy"). Parece que al final acabarán coexistiendo ambas, la primera para usos empresariales y de comercio electrónico, y la segunda para uso particular.

2.3 Seguridad en las comunicaciones de correo electrónico.

En un principio, el correo electrónico es una aplicación insegura. Esta inseguridad se puede ver reflejada en los distintos puntos que se han estudiado con anterioridad, y que ahora pasamos a comentar:

- ***Webmail*:** en una conexión no segura de un servidor *Webmail* (la dirección requerida es http:// y no https://) toda la información, incluidos nombre de usuario y contraseña, no está encriptada pasando de forma clara entre nuestra computadora y el servidor de *Webmail.*

- ***SMTP:*** el protocolo SMTP no encripta los correos electrónicos; la comunicación existente entre los servidores SMTP, que se encargan de llevar un correo a su destino, se hace de forma clara, por lo que cualquiera puede tener acceso a ella. Por otra parte, un servidor de correo podría requerir el envío de las credenciales de un usuario para poder comunicarse con otros servidores SMTP con objeto de buscar caminos alternativos para el envío del mensaje, de nuevo este envío de usuario y contraseña se llevaría a cabo de forma clara por la red. Finalmente, los mensajes envía vía SMTP incluyen información sobre la máquina que envió el correo y el programa de correo usado para ello, esta información es, una vez más, accesible para todos los receptores del correo.

- ***POP* e *IMAP:*** como hemos visto en las secciones anteriores los protocolos POP e IMAP requieren el envío del nombre de usuario y contraseña para autenticarse ante el servidor de correo, estas credenciales no están encriptadas, por lo que cualquier programa *sniffer* tiene acceso al flujo de información entre un ordenador personal y el proveedor del servicio de correo.

- ***BackUps:*** los mensajes de correo electrónico son almacenados en los servidor SMTP en texto plano, por lo que los administradores de estas máquinas pueden leerlos. Además estos mensajes son guardados durante un tiempo ilimitado sin ningún tipo de seguridad, por que cualquier persona puede tener acceso a ellos.

Se han expuesto un pequeño número de vulnerabilidades que afectan al *email*. En los siguientes apartados se ahondará con mayor profundidad en estos problemas de seguridad y cuales pueden ser sus posibles soluciones.

2.3.1 Problemas de seguridad en una comunicación de correo electrónico.

En este apartado, se describen muchos de los problemas de seguridad más comunes presentes en una comunicación de correo electrónico.

- ***Escuchas Secretas:*** Internet es un "lugar" transitado por muchas personas. Es muy fácil para alguien que tiene acceso a los ordenadores o las redes capturar y leer la información que está viajando por la red.

- ***Robo de Indentidad:*** si alguien obtiene el nombre de usuario y la contraseña que una persona usa para acceder a su servidor de correo, puede leer los correos y enviar nuevos mensajes falsos haciéndose pasar por el usuario suplantado.

- ***Invasión de la Privacidad:*** se puede tener acceso a la dirección IP de cualquier cliente de correo, con la cual se podría obtener fácilmente el lugar de residencia de un usuario, lo que supone alteración de la privacidad.

- ***Alteración de los correos:*** El administrador de los servidores SMTP que son usados en el camino que recorre un correo, puede tener permisos para acceder a dichos mensajes, leerlos, borrarlos, o modificarlos antes de que continúen hacia su destino final. No hay mecanismos para saber si un mensaje ha sido borrado o modificado intencionadamente, lo que supone un gran perjuicio para los usuarios.

- ***Mensajes Falsos:*** es muy fácil redactar mensajes que parezcan enviados por una persona, los virus están basados en este sistema; en general, no hay forma de saber si el aparente remitente del mensaje, es el verdadero emisor, ya que su nombre se usuario puede ser fácilmente falsificado.

- ***Réplicas de Mensajes:*** como hemos visto, un mensaje puede ser guardado, modificado o reenviado con posterioridad; cualquier persona puede recibir un mensaje válido y original, pero puede ocurrir que los siguientes estén alterados, provocando en el usuario la creencia de que todos son válidos.

- ***Almacenamientos sin Protección:*** en la introducción se dijo que los mensajes almacenados en los servidores SMTP lo están de forma clara, por lo que las copias de dichos mensajes también lo están. Estas copias pueden estar años almacenadas por lo que pueden ser leídas por cualquier usuario que tenga acceso a ellas.

- ***Repudio:*** ya que los mensajes pueden ser falsificados, no hay forma de saber si alguien en particular envió un mensaje; esto significa que si alguien envió un mensaje en nombre de otra persona también puede hacer que lo rechacen.

2.3.2 Encriptación simétrica y asimétrica.

El siguiente apartado pretende dar a conocer alguno de los métodos utilizados para intentar prevenir las situaciones anteriores. En concreto se tratarán dos tipos de encriptaciones, la simétrica y la asimétrica.

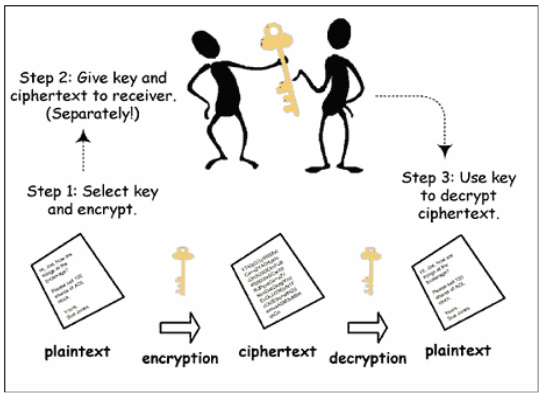


Figura 15: Encriptación simétrica y asimétrica.

Clave de encriptación Simétrica

En la encriptación con clave simétrica ambos usuarios comparten una clave “secreta”. Usando esta clave, se puede encriptar un mensaje en *cyphertext*, que es una secuencia de caracteres aleatoria y es completamente imposible que alguien pueda descifrarla a menos que tenga la clave secreta, en cuyo caso, puede desencriptar el *cyphertext* y obtener el mensaje original, con los que puede leerlo.

Mediante este método, los problemas anteriores de escuchas secretas y almacenamientos sin protección ya no son un problema, siendo muy difícil la modificación del mensaje en tránsito.

El único problema de este sistema es que ambas partes deben compartir la clave secreta, y a menos que la persona se encuentre en un lugar próximo, no se garantiza que el mensaje con la clave no sea interceptado.

Códigos de autenticación de mensajes

Un código de autenticación de mensajes no es más que la generación de una serie de caracteres llamados “huella” a partir de un mensaje. Cualquier cambio en un mensaje producirá una “huella” diferente. Es imposible obtener un mensaje original a partir de esta “huella” o encontrar dos mensajes que tenga la misma “huella”.

Estos códigos son muy útiles para comprobar si un mensaje ha sido alterado; si un usuario tiene la “huella” original y la compara con la “huella” que le ha llegado con el mensaje puede comprobar fácilmente si el mensaje ha sufrido alguna modificación.

Clave de encriptación asimétrica

También es conocida como encriptación con clave pública, donde cada usuario tiene dos claves. Cualquier *cyphertext* creado usando una de las claves son puede ser descifrado por la otra clave. Este sistema es diferente al de la clave pública simétrica visto anteriormente, donde una sola clave llevaba a cabo las dos funciones.

En este sistema, las dos claves que posee cada persona se denominan claves “pública” y “privada”. La clave “pública” la puede obtener cualquier persona que quiera una copia, mientras que la privada se debe mantener en secreto. La seguridad se este método sólo depende de que la clave privada se mantenga oculta.

La clave de encriptación asimétrica, permite realizar diversas cosas:

- **Enviar un mensaje encriptado:** para enviar un mensaje a alguien, lo que hay que hacer es cifrarlo con dicha clave pública, y sólo el receptor del mensaje que posea la clave privada podrá descifrar y leer dicho mensaje. Esto solventa el problema de las escuchas secretas y el problema del envío de claves secretas que hay que hacer en el caso simétrico.

- **Demostrar quien envió un mensaje:** para demostrar a cualquier usuario que una determinada persona envió un mensaje, ésta debe encriptar el mensaje con su clave privada, lo que da lugar a que cualquiera con una clave pública pueda descifrarlo y leer su contenido; el hecho de que la clave pública desencripta el mensaje, prueba que sólo la persona que lo envió pudo enviarlo.

- **Firmar un mensaje:** la firma de un mensaje prueba que un determinado usuario envió un mensaje y permite a su receptor comprobar si el mensaje fue modificado durante su camino. Esta se hace usando la clave privada para encriptar un “resumen” del mensaje al tiempo que se envía. El receptor puede descifrar este “resumen” y compararlo con los “resúmenes” de otros mensajes recibidos; si coinciden, el mensaje no ha sido alterado y fue enviado por el usuario real.

- **Mensajes encriptadas y firmados:** la forma más segura de realizar una comunicación es añadir primero una firma al mensaje y posteriormente cifrarlo junto con la firma mediante la clave pública. Esto combina todos los beneficios de todas las técnicas vistas hasta ahora: seguridad frente a escuchas y almacenajes indeseados, falsos transmisores e integridad de los mensajes

2.3.4 Utilización de SSL y TLS en una comunicación de correo electrónico

Una forma fácil de hacer que el correo electrónico sea más seguro es usar un proveedor de correo que soporte SSL ("*Secure Socket Layer*") para sus servidores de correo *web*, POP, IMAP y SMTP. TLS es un tipo de específico SSL que puede ser iniciado al comienzo de una sesión de correo, mientras que SSL debe ser iniciado antes de enviar un *email*.

SSL es una combinación de mecanismos de encriptación simétricos y asimétricos. Cuando se desea conectar a un servidor usando SSL, ocurren una serie de cosas:

- El servidor usa su clave privada para comprobar que, efectivamente, es el servidor al que el usuario se quiere conectar; lo que permite al usuario asegurarse de que no se conecta a un "usuario intermedio" que está intentando interceptar la comunicación.

- El usuario envía al servidor su clave pública.

- El servidor genera una "clave secreta" y se la envía al usuario cifrada usando la clave pública que este último envió.

- El usuario y el servidor se comunican usando la encriptación de clave simétrica mediante esta clave secreta comparatida (la clave de encriptación simétrica es más rápida que la asimétrica).

Los beneficios de SSL son dos:

- El usuario puede determinar si se está conectando al servidor correcto.
- La comunicación entre el usuario y el servidor se realiza de forma segura.

Si al conectarse al servidor usando SSL, el usuario recibe cualquier mensaje de aviso, eso puede ser indicativo de que la comunicación está siendo interceptada. Estos avisos indican una de las siguientes circunstancias:

- El certificado SSL del servidor (por ej. par de clave pública/privada) ha expirado.

- Alguna de la información presente en el certificado no concuerda con la información esperada (por ej. El certificado contiene el nombre de un servidor distinto al que el usuario está intentando conectarse).

- El certificado fue generado por una autoridad insegura.

Usando SSL para *Webmail*, POP, IMAP y SMTP se asegura que la comunicación entre el ordenador personal de un usuario y su proveedor de correo estará encriptada. El contenido del mensaje, nombre de usuario, contraseña, estará escondida a posibles *hackers*; sin embargo SSL no asegura la protección de los correos una vez han dejado el servidor SMTP hacia sus destinos; lo cual quiere decir que realmente lo único que se protege son los nombres de usuario y contraseña, pero no el contenido del mensaje.

2.3.5 Privacidad mediante SMTP Anónimo.

En las secciones anteriores se vió que cuando un usuario envía un correo mediante un cliente de correo, no un *Webmail*, la dirección IP de la máquina que envió el correo se incluye en el mensaje, la cual puede ser vista por todos los receptores de dicho mensaje.

Dependiendo de la privacidad ofrecida por el Proveedor de Servicios de Internet y el tipo de servicio y conexión proporcionado, esta información puede ser usada para determinar la zona del país en que se encuentra el usuario, la ciudad en la que vive e incluso la dirección de su domicilio.

Los servicios de SMTP Anónimo proporcionan la misma funcionalidad que los servidores SMTP normales, aumentando la privacidad del usuario. Estos servicios, normalmente reciben el mensaje vía SMTP Autenticado y rastrean el mensaje borrando toda la información referente a la dirección IP de la máquina, el programa de correo usado y otras informaciones no relevantes; posteriormente, estos mensajes siguen su camino hasta su destinatario.

El resultado final es que los usuarios reciben el mensaje igual que si no se hubiera utilizado el servicio *Anónimo*, excepto que los mensajes de respuesta deben volver a pasar por el servidor Anónimo. Estos servidores saben quien es cada usuario basándose únicamente en la dirección de correo y en el contenido del mensaje, pero no saben desde donde se envía en correo ni el programa usado para ello.

2.3.6 El correo electrónico y las claves de cifrado asimétrico (PGP y S/MIME)

Como hemos visto, SSL protege contraseñas y el contenido del menaje, pero no solventa otra tipo de problemas como pueden ser: el repudio, modificación de mensajes, encriptación,… Esto es así porque SSL solo garantiza la protección entre el usuario y el servidor SMTP, deteniéndose en ese punto. Incluso con SSL, los mensajes son almacenados en el servidor SMTP en texto claro.

Hay dos formas de encriptación con clave asimétrica para el *email*: S/MIME y PGP; ambas formas permiten añadir firma y cifrado a los mensajes.

Problemas de Interoperabilidad

PGP y S/MIME solventan muchos de los problemas vistos hasta el momento pero, por el contrario, crean otro: la interoperabilidad. Ambos son totalmente incompatibles, si un usuario usa PGP y otro S/MIME los mensajes no serán transmitidos de forma segura.
PGP ha sido un estándar de Internet (OpenPGP-RFC 2440) desde 1997 y ha encriptado cerca del 90% del tráfico de correos electrónicos que circula por la red, por lo tanto, es compatible con la mayoría de los clientes de correo; de todas formas, muchos de estos clientes de correo dan la posibilidad de elegir entre uno u otro sistema de encriptación.

El otro problema de interoperabilidad es el intercambio de claves. Si un usuario desea enviar a otro un mensaje cifrado, en primer lugar necesita su clave pública, por lo que es necesario intercambiar dichas claves antes de que la comunicación empiece. PGP ofrece "servidores de claves" desde donde los usuarios pueden descargarse las claves y el proceso es más sencillo. Sin embargo, no todos los usuarios tienen sus claves almacenadas en un servidor de claves y no todos usan PGP, por lo que el intercambio de claves es un impedimento para el envío de mensajes seguros.

2.3.7 Conclusiones

Como se decía al comienzo de este apartado el correo electrónico es totalmente inseguro aunque, como hemos visto, hay mecanismos para intentar minimizar los problemas de privacidad que puedan surgir.

Es muy recomendable utilizar SSL en todas nuestras comunicaciones, a pesar de que sólo protege el camino entre el usuario y el servidor SMTP y, si además se pueden emplear otros mecanismos como el uso de servidores SMTP Anónimos o sistemas de encriptado de correos como PGP ó S/MIME, se estará alcanzando un alto grado de seguridad.

Capítulo 3: DESARROLLO DE LA APLICACIÓN

3. Desarrollo de la aplicación.

3.1 Descripción general de la solución implementada.

Este trabajo versa acerca de una aplicación cliente-servidor usando clientes de correo electrónico para la transferencia de datos; en concreto, se ha optado por utilizar como cliente de correo el proveedor *Gmail*.

El esquema general de la aplicación se muestra a continuación:

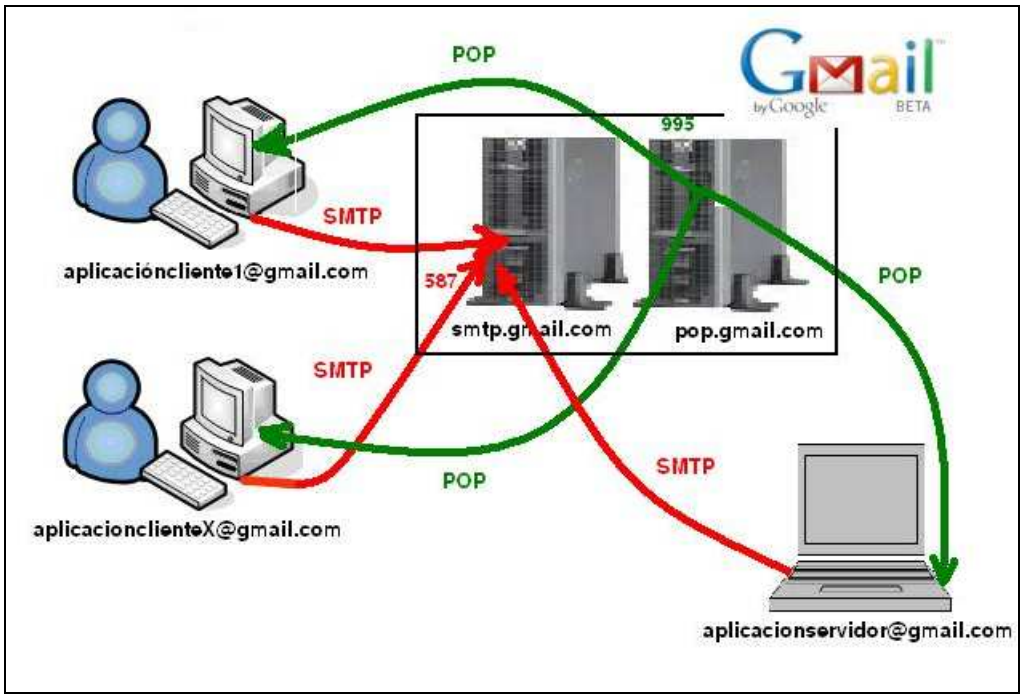


Figura 16: Esquema de la aplicación a desarrollar.

- Como descripción general de su funcionamiento, nos encontramos con una serie de aplicaciones llamadas “Clientes” que son las encargadas de recoger datos provenientes del exterior, de los usuarios o equipos conectados a ellas. Estos datos son encapsulados en un correo electrónico, el cual es enviado al servidor de correo de *Gmail*.

- Por otra parte, tenemos una aplicación llamada “Servidor”, también conectada al servidor de correo de *Gmail*, que se encarga de recoger los correos nuevos, que llegan a éste último, para su posterior procesamiento.

- Una vez procesados los correos por parte de la aplicación “Servidor”, son devueltos al servidor de *Gmail*, donde la aplicación *Cliente* se los descargará, obteniendo de esta manera la respuesta a la petición realizada.

3.2 Seguridad en la solución implementada.

Como hemos visto en apartados anteriores, la seguridad es un tema crítico a tener en cuenta en nuestra aplicación, por lo que antes de pasar a explicar con más detalle la solución propuesta a este trabajo, se va a realizar un inciso en este tema.

- Un servidor SMTP normal trabaja a través del puerto 25, siendo sólo necesario tener unas credenciales válidas en el mismo que permitan enviar correo. Lo habitual en una aplicación cualquiera es usar el servidor SMTP de nuestra cuenta de correo favorito con esta técnica sin problemas.
- Sin embargo, si nuestra cuenta de correo es de *Gmail*, como ocurre en este caso, la técnica convencional no funciona. El motivo es que *Gmail* usa un puerto diferente (587) y además precisa el uso de una conexión cifrada con SSL para seguridad. Como veremos más adelante, la clase *SmtpClient* ayuda a solucionar este inconveniente, ya que básicamente le indicaremos un par de parámetros extra para indicar el puerto a utilizar y la obligatoriedad de SSL.

- En cuanto al protocolo POP3, hemos visto que en el estado de Autorización, se usan los comando *USER* y *PASS* para enviar el nombre de usuario y contraseña respectivamente en texto plano, por lo que puede ser capturado por cualquier programa *sniffer*. Este problema, lo solucionan en parte otros comandos menos utilizados, siempre y cuando sean soportados por el proveedor de correo:

    - *APOP* con autenticación MDS. En este protocolo, el cliente de correo envía un hash codificado de la contraseña al servidor, en lugar de enviar la contraseña en texto plano.

    - *KPOP* con autenticación Kerberos.

    - *RPOP* con autenticación RPOP, que utiliza un identificador de usuario similar a una contraseña para autenticar las peticiones del protocolo POP. No obstante, este identificador o ID no está codificado, de modo que RPOP no es más seguro que el protocolo POP normal.

- Sin embargo, de nuevo *Gmail* nos obliga a crear una conexión segura para llevar a cabo la comunicación con su servidor, en este caso en el puerto 995. Para crear esta conexión, se va a hacer uso de la clase *SslStream*, cuyos principales métodos y propiedades se explican en el siguiente punto.

3.2.1 Clase SslStream.

Esta clase proporciona una secuencia utilizada para comunicaciones cliente y servidor que usa el protocolo de seguridad Capa de Sockets Seguros (SSL) para autenticar el servidor y, opcionalmente, el cliente. Para ello, al comienzo de cada proyecto, se deben añadir las siguientes librerías:

```
System.Net.Security;
System.Security.Authentication;
System.Security.Cryptography.X509Certificates;
```

Los protocolos SSL ayudan a proporcionar comprobaciones de confidencialidad e integridad para mensajes transmitidos mediante una *SslStream*. Una conexión SSL, como la proporcionada por *SslStream*, se debe usar al transmitir información confidencial entre un cliente y un servidor. Si se usa una *SslStream*, se puede evitar que alguien lea y manipule información mientras está pasando por la red.

Una instancia de *SslStream* transmite datos mediante una secuencia proporcionada al crear la *SslStream*. Al proporcionar esta secuencia subyacente, puede optar por especificar si al cerrar la *SslStream*, también se cierra la secuencia subyacente. Normalmente, la clase *SslStream* se usa con las clases *TcpClient* y *TcpListener*. El método GetStream proporciona una NetworkStream adecuada para su uso con la clase *SslStream*.

Después de crear una *SslStream*, se deben autenticar el servidor y, opcionalmente, el cliente. El servidor debe proporcionar un certificado X509 que establezca la prueba de su identidad y puede solicitar que también lo haga el cliente. La autenticación se debe llevar a cabo antes de transmitir información mediante una *SslStream*. Los clientes inician la autenticación mediante los métodos *AuthenticateAsClient* sincrónicos, que se bloquean hasta que concluye la autenticación o los métodos *BeginAuthenticateAsClient* asincrónicos, que no se bloquean a la espera de que concluya la autenticación. Los servidores inician la autenticación mediante los métodos AuthenticateAsServer sincrónicos o *BeginAuthenticateAsServer* asincrónicos. Tanto el cliente como el servidor deben iniciar la autenticación.

La autenticación la controla el proveedor de canales de la interfaz de proveedor de soporte de seguridad (SSPI). Se da al cliente la oportunidad de controlar la validación del certificado del servidor especificando un delegado *RemoteCertificateValidationCallback* al crear un *SslStream*. El servidor también controla la validación proporcionando un delegado de *CertificateValidationCallback*. El método al que hace referencia el delegado incluye el certificado de la parte remota y los errores SSPI encontrados mientras se validaba el certificado. Observe que si el servidor especifica un delegado, el método de éste se invoca independientemente de si el servidor solicitó autenticación del cliente. Si no lo hizo, el método del delegado del servidor recibe un certificado nulo y una matriz vacía de los errores del certificado.

Si el servidor requiere autenticación del cliente, éste debe especificar uno o varios certificados con fines de autenticación. Si el cliente tiene más de un certificado, puede proporcionar un delegado *LocalCertificateSelectionCallback* para seleccionar el certificado correcto para el servidor. El certificado del cliente debe estar en el almacén de certificados "My" del usuario actual. No se admite la autenticación del cliente mediante certificados para el protocolo Ssl2 (SSL versión 2).

Si en la autenticación se produce un error, se recibe una excepción *AuthenticationException* y el *SslStream* ya no se puede usar. Debe cerrar este objeto y quitar todas las referencias al mismo para que lo pueda recoger el recolector de elementos no utilizados.

Cuando es correcto el proceso de autenticación, también denominado protocolo de enlace SSL, se establece la identidad del servidor (y opcionalmente la del cliente), y tanto el cliente como el servidor pueden utilizar la *SslStream* para intercambiar mensajes. Antes de enviar o recibir información, el cliente y el servidor deben comprobar los servicios y niveles de seguridad proporcionados por la *SslStream* con el fin de determinar si el protocolo, los algoritmos y las claves de seguridad satisfacen los requisitos de integridad y confidencialidad. Si no basta con la configuración actual, se debe cerrar la secuencia. Puede comprobar los servicios de seguridad proporcionados por *SslStream* mediante las propiedades *IsEncrypted* y IsSigned.

En el Anexo I, se puede encontrar una descripción detallada de los métodos y propiedades más relevantes de esta clase.

3.3 Obtención de parámetros a través de archivos XML.

Antes de pasar a describir detalladamente las capas que componen el sistema, se va a mostrar la forma implementada en este proyecto para la obtención de algunos datos relevantes para el correcto funcionamiento de las aplicaciones.

Las aplicaciones *Cliente* y *Servidor*, necesitan una serie de parámetros para acceder a las cuentas de correo, como son las direcciones y los puertos de los servidores SMTP y POP3 de *Gmail*, las cuentas y contraseñas de usuario, etc. Por ello, se han implementado unos archivos escritos en lenguaje XML externos a la aplicación que permiten recoger y variar dichos parámetros sin necesidad de recompilar el código.

Un aspecto muy importante a considerar respecto a estos archivos, es que para acceder a ellos desde las diferentes aplicaciones, se hace teniendo en cuenta una ruta relativa (`@"..\..\ArchivosConfiguracion\Parametros_Configurables.xml"`). Por lo tanto, siempre deberemos mantener la carpeta que contiene dichos archivos, en este caso *ArchivosConfigurables*, junto a la carpeta que contiene los archivos ejecutables de la arquitectura. Gracias a ello, se dota de portabilidad al proyecto, pudiendo ejecutarlo en cualquier lugar.

3.3.1 Archivos implementados en el proyecto.

3.3.3.1 Archivo “Parámetros Configurables”.

En este archivo se encuentran los parámetros necesarios para la configuración de los protocolos implicados en las diferentes aplicaciones. En capítulos posteriores se explica con más detalle su contenido y configuración. Un ejemplo de este tipo de archivo se muestra seguidamente:

```
<?xml version="1.0" encoding="utf-8" ?>
<protocolos>
      <smtp>
            <servidor>smtp.Gmail.com</servidor>
            <origen>aplicacioncliente1@Gmail.com</origen>
            <destino> aplicacionservidor@Gmail.com </destino>
            <puerto>587</puerto>
            <contrasena>upct2007</contrasena>
      </smtp>
      <pop3>
            <servidor>pop.Gmail.com</servidor>
            <puerto>995</puerto>
            <usuario> aplicacioncliente1@Gmail.com </usuario>
            <contrasena>upct2007</contrasena>
      </pop3>
</protocolos>
```

Figura 17: Archivo "Parámetros Configurables"

3.3.3.2 Archivo "Tipos de Mensaje".

Este archivo contendrá aquellos tipos de mensaje que se van a analizar, y de cuyo contenido depende el funcionamiento del programa de una forma u otra. Como se puede observar en el siguiente código, cada mensaje consta de un tipo que define el mensaje implementado, un asunto que nos informa del correo a enviar o recibir y un texto que define el cuerpo del mensaje y que podrá cambiar en función de las necesidades.

De nuevo, más adelante se explicará cómo configurar este fichero, ahora, se muestra su contenido:

```
<?xml version="1.0" encoding="utf-8" ?>
<mensajes>
      <mensaje>
            <tipo>ECHO</tipo>
            <asunto_esperado>Echo Request</asunto_esperado>
            <asunto_respuesta>Echo Reply</asunto_respuesta>
            <texto>Echo Reply del mensaje recibido</texto>
      </mensaje>
</mensajes>
```

Figura 18: Archivo "Tipos de Mensaje".

3.4 Descripción detallada de cada una de las partes del sistema.

3.4.1 Aplicación Cliente.

Como se ha descrito anteriormente, de una forma resumida, cada una de las aplicaciones *Cliente* se encarga de la recogida de datos provenientes del exterior. Estos datos, una vez formateados, forman un mensaje de correo electrónico que es enviado al servidor de correo de *Gmail*. Dicho correo será descargado y procesado por la aplicación "Servidor", que enviará su respuesta de nuevo al servidor, donde la aplicación cliente procederá a su descarga y análisis. Para dicha descarga, en el *Cliente* se implementa un *timer* que continuamente se encuentra chequeando la llegada de nuevos correos.

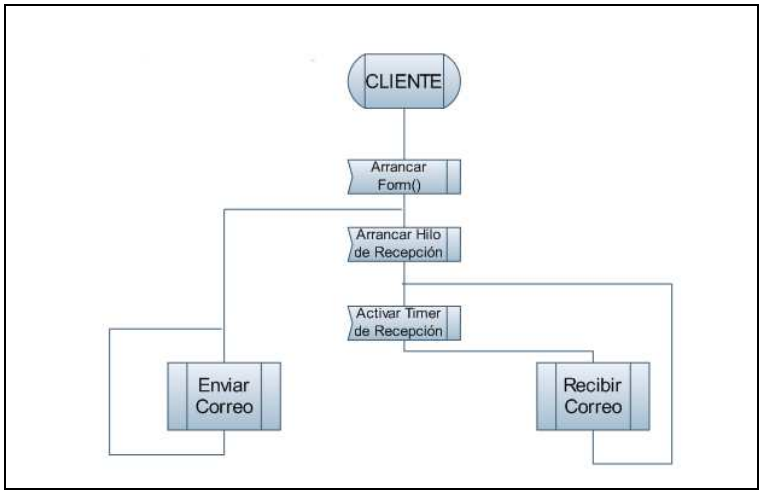


Figura 19: Diagrama de estados principal de la aplicación *Cliente.*

Tal y como se ha podido observar en el apartado anterior, tanto la aplicación *Cliente* como la aplicación *Servidor* se encuentran divididas en diferentes capas, que implementan diversas funcionalidades, abstrayendo tanto a las capas inferiores como superiores de su implementación. A continuación, se describen dichas capas para la aplicación *Cliente*:

3.4.4.1 Capa de Comunicaciones Cliente.

Esta capa es común tanto a la aplicación *Cliente* como a la aplicación *Servidor*. Es la encargada de gestionar el envío y la recepción de correos con el servidor de correos de *Gmail*.

A su vez, para una mejor gestión de las funcionalidades, esta capa está subdividida en una Subcapa de Comunicaciones Transmisora, para el envió de correos al servidor de *Gmail*, y una Subcapa de Comunicaciones Receptora, encargada de la recepción de los correos procedentes de *Gmail*.

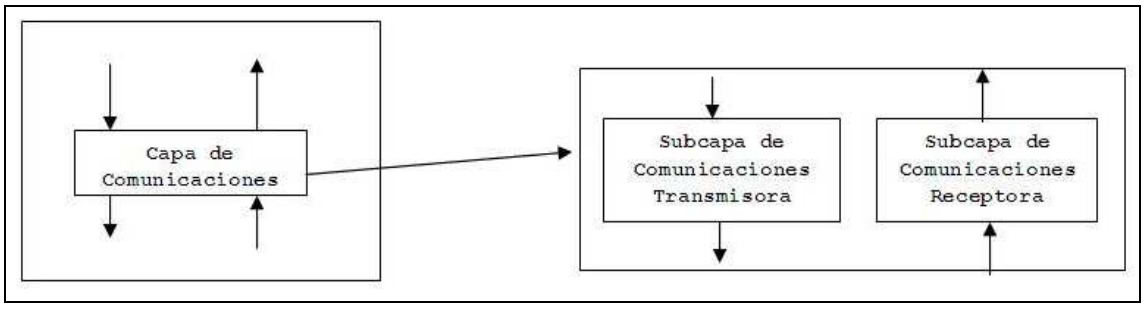


Figura 20: Capa de Comunicaciones Cliente.

3.4.4.1.1 Subcapa de Comunicaciones Transmisora.

Subcapa encargada de recibir el correo electrónico ya formado proveniente de la Capa de Procesamiento, para su posterior envío al servidor de correo de *Gmail*.

Esta subcapa es la encargada de crear el objeto cliente de correo, que servirá para comunicar al servidor de correo de *Gmail* con nuestra aplicación *Cliente*. Para ello, debe enviar las credenciales de autenticación al servidor (usuario y contraseña), que han sido obtenidas del archivo *Parámetros_Configurables.xml*; todo ello sobre un canal seguro sobre SSL, como exige el servidor de *Gmail*.

Además de los parámetros necesarios para crear la conexión, se han creado dos nuevas cabeceras, que serán añadidas cuando el correo electrónico se envíe. Estas cabeceras son:

- "ID-Usuario:", implementada para poder discernir a qué cliente va destinado un mensaje de respuesta que se encuentra en el servidor de *Gmail*. En este caso, al ser una aplicación multicliente, donde los clientes no se conocen entre sí, se ha decidido usar como identificador de usuario la propia dirección de correo electrónico del transmisor del mensaje, de esta forma nos aseguramos que el ID es único para cada *Cliente*.

- "ID:", que sirve para añadir un identificador a cada mensaje enviado, y así poder asociar consultas y respuestas. Para generar estos identificadores, que son obligatoriamente cadenas de texto ya que las cabeceras no admiten otro formato, se ha creado una clase *RandomID*, que genera aleatoriamente cadenas de caracteres alfabéticos de 12 letras de longitud, lo que es suficiente para que no se generen dos cadenas iguales durantes los envíos de los correos. Por último, comentar que los IDs generados son almacenados en un *ArrayList()*, para poder comprobar que los mensajes que se reciben, efectivamente, fueron enviados por un cliente determinado.

Todo lo descrito con anterioridad se muestra en el siguiente flujograma para una mejor comprensión:

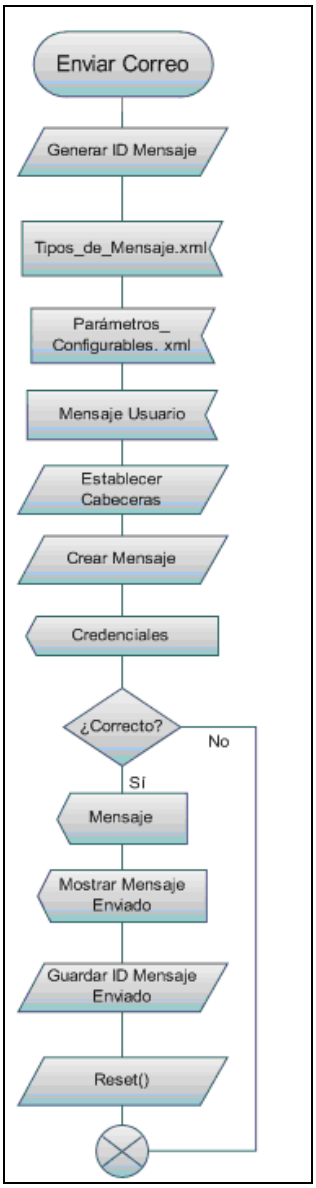


Figura 21: Diagrama de flujo Subcapa de Comunicaciones Transmisora.

El protocolo SMTP, que es el estándar empleado para la transmisión de mensajes de correo electrónico, es implementado por esta capa. Para la implementación de dicho protocolo, se han hecho uso de una serie de clases de la librería *System.Net.Mail*, disponible en el API de C#. En concreto para esta implementación se ha hecho uso de la clase *SmtpClient* encargada del envió de un correo electrónico a un servidor SMTP para su entrega; la descripción detallada de los métodos y propiedades más relevantes de dicha clase se puede consultar en el Anexo II.

3.4.4.1.2 Subcapa de Comunicaciones Receptora.

Esta subcapa es la encargada de recibir los mensajes provenientes del servidor de correo de *Gmail*, para su posterior envío a la capa superior de procesamiento de cada una de las aplicaciones.

Para ello, dicha capa debe estar chequeando continuamente la llegada de nuevos correos al servidor de correo de *Gmail*. Dicho chequeo se ha implementado a través de un *Timer*, que se reinicia cada "x" segundos.

Como se ha descrito en capítulos anteriores, el protocolo POP3 se basa en una serie de comandos que intercambian el cliente y el servidor de correo. Pero para ello, antes se ha de crear una conexión entre ambos equipos; en este caso, la conexión se ha creado sobre un canal seguro, a través del protocolo SSL, ya que el servidor de *Gmail* sólo funciona de este modo.

```
pop3Stream = new SslStream(cliente.GetStream(), false);
((SslStream)pop3Stream).AuthenticateAsClient(servidorPOP3);
```

Una vez ha sido creada la conexión, el servidor de *Gmail* envía un saludo indicando que la comunicación se ha realizado con éxito y que está preparado para la recepción de peticiones.

Recibido el saludo por parte del servidor, el cliente entra en el estado de Autorización, donde envía sus credenciales, *"USER"* y *"PASS"* al servidor. Si no se ha producido ningún error, se pasa al estado de Transacción.

En el estado de Transacción, el cliente envía diversos comandos (*"STAT"*, *"LIST"*,…) que le proporcionan diversa información acerca del número de mensajes en el servidor, su tamaño,…

En el caso de que haya mensajes presentes en el servidor de *Gmail*, pasaremos a la Capa de Procesamiento, que será la encargada de descargar el correo para extraer los parámetros necesarios para su posterior procesamiento. En el caso de que no haya mensajes presentes en el servidor, ese esperará a que venza el *Timer* para volver a comprobar la llegada de mensajes.

El siguiente flujograma muestra todo el proceso de envío de comandos:

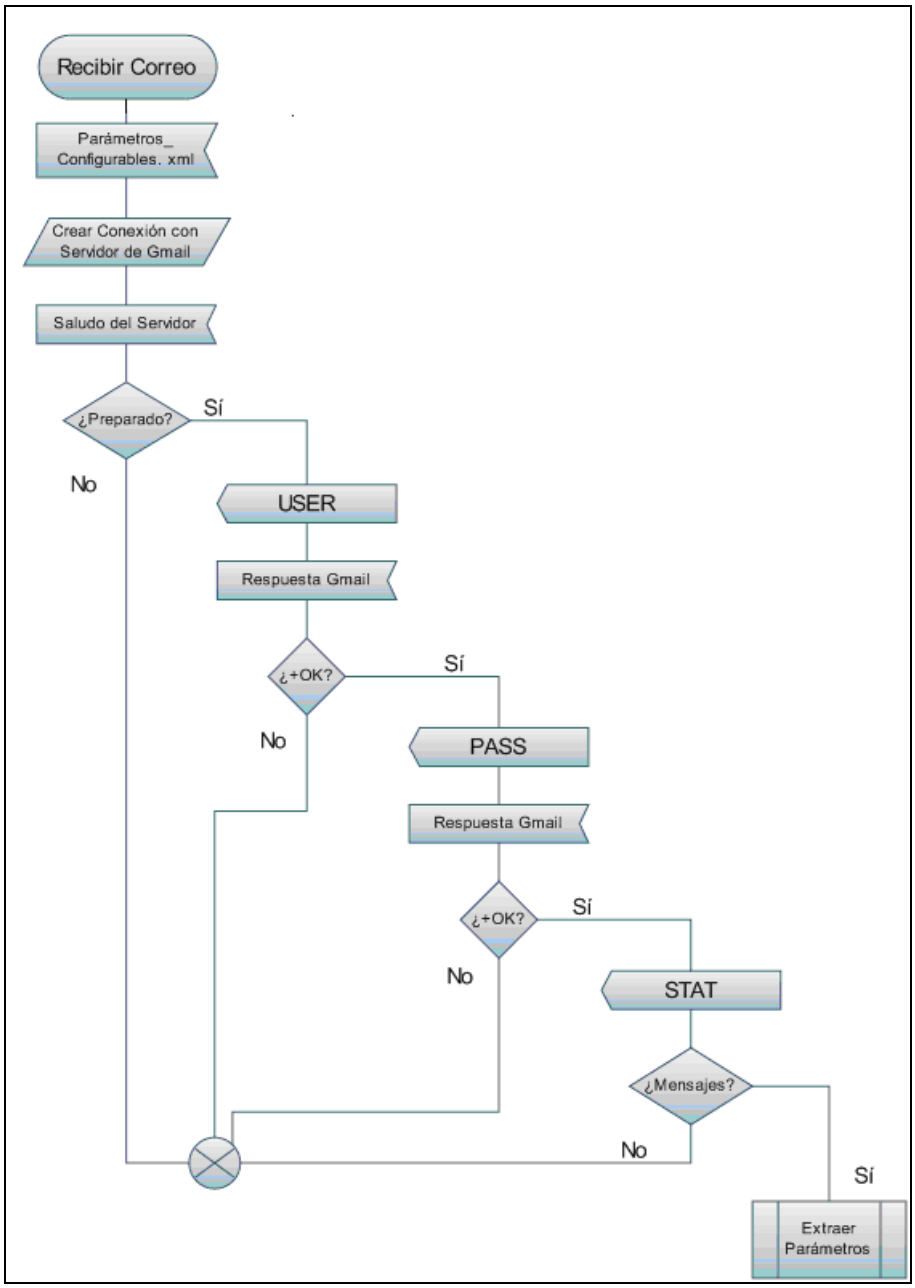


Figura 22: Flujograma Subcapa de Comunicaciones Transmisora.

Capa de Procesamiento Cliente.

Esta capa se encuentra entre la Capa de Comunicaciones anteriormente descrita y la capa de aplicación. A su vez, esta capa estará dividida en dos subcapas, denominadas Subcapa de Procesamiento Transmisora, encargada de fijar todos los parámetros necesarios para la composición del correo electrónico a enviar; y Capa de Procesamiento Receptora, cuya función es procesar los mensajes de correo electrónico en función de los parámetros buscados.

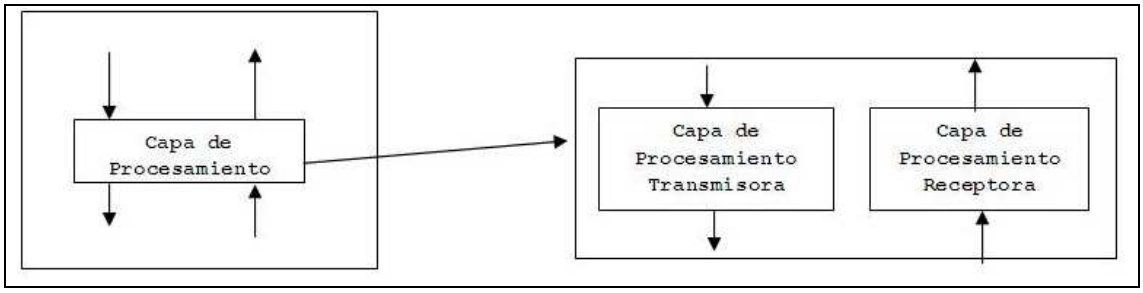


Figura 23: Capa de Procesamiento de la aplicación *Cliente.*

3.4.4.2.1 Subcapa de Procesamiento Transmisora.

Esta subcapa es la encargada de proporcionar todos los datos necesarios para enviar un correo electrónico: dirección origen, dirección destino, asunto, cuerpo del mensaje, archivos adjuntos, etc.

Las clases más relevantes para esta subcapa, que son proporciondas por la API de C#, son la clase *MailAddress* y la clase *MailMessage*, cuyas descripción se encuentra en los Anexos III y IV respectivamente.

3.4.4.2.2 Subcapa de Procesamiento Receptora.

Una vez comprobada la existencia de mensajes en el servidor de correo de *Gmail*, esta subcapa se encargará de su descarga y procesamiento.

En primer lugar, para cada uno de los mensajes presentes en el servidor, se envía el comando *"RETR"*, que nos devuelve el mensaje completo, incluidas las cabeceras, en una cadena de texto.

Posteriormente, se ha implementado un método llamado *ExtraerParametros()* que se encarga de formatear dicha cadena de texto, para extraer tanto el cuerpo como las cabeceras que son necesarias para el análisis del correo; en el caso de las aplicaciones *Cliente*, es necesario extraer las cabeceras *"FROM:"*, *"ID-Usuario:"*,*"ID:"* y *"Subject:"*.

A continuación, dichos parámetros se pasan a otro método llamado *Procesar()*, que será explicado con posterioridad, que se encarga de realizar una serie de comprobaciones para ver si el mensaje recibido pertenece a dicho cliente. Si es así, el mensaje es borrado del servidor de correo de *Gmail* mediante el envío del comando *"DELE"*; para que

dicho borrado se lleve a cabo definitivamente, hay que enviar el comando *"QUIT"*, que cierra la conexión, ello se lleva a cabo mediante el método *Desconectar()*.

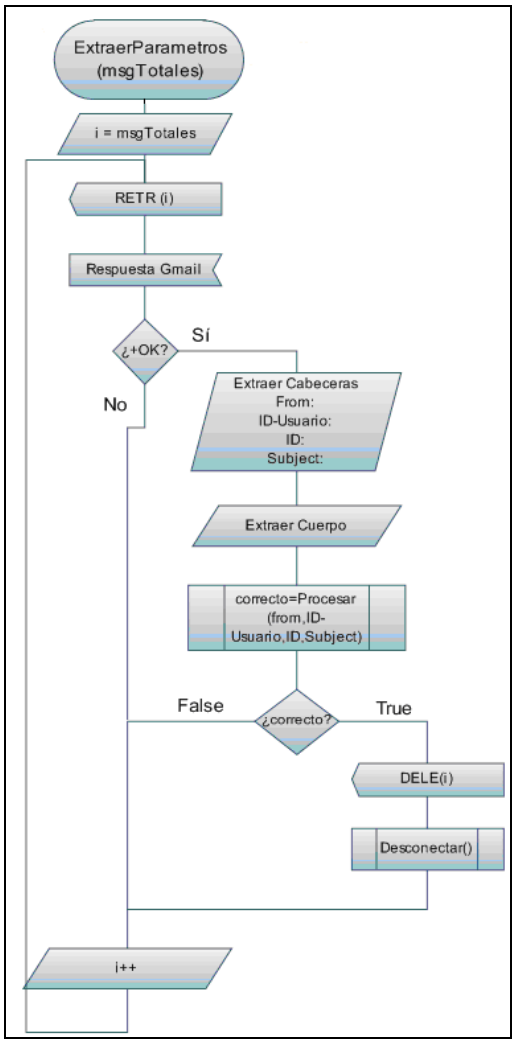


Figura 24: Flujograma Subcapa de Procesamiento Receptora de la aplicación *Cliente.*

En último lugar, se describe el método *Procesar()* al cual se le pasa como argumento los parámetros de las cabeceras extraídos anteriormente, devolviendo como resultado una variable booleana que indica si el correo electrónico es el esperado o no. Las comprobaciones que se realizan son las siguientes:

1º) Comprobamos si el mensaje recibido proviene de la aplicación *Servidor*, que es la encargada de procesar el mensaje enviado y suministrar la respuesta a dicho mensaje; con esta comprobación evitamos descargar mensajes provenientes de otros clientes.

2º) Se comprueba, mediante el parámetro extraído de la cabecera *"ID-Usuario:"*, que el mensaje va dirigido a nosotros.

3º) Mediante el asunto del correo, suministrado por la cabecera *"Subject:"*, comprobamos que el asunto recibido se corresponde con el asunto enviado. En nuestro caso, como la aplicación *Cliente* envió un correo con asunto *EchoRequest*, esperamos recibir un correo con asunto *EchoReply.*

4º) Para finalizar, mediante el identificador de mensaje proporcionado por la cabecera *"ID:"*, llamamos al método *ProcesarID()* que recorre el *ArrayList()* que contiene los IDs de los correos enviados, para asegurarnos que la respuesta recibida se corresponde con un mensaje todavía sin confirmar.

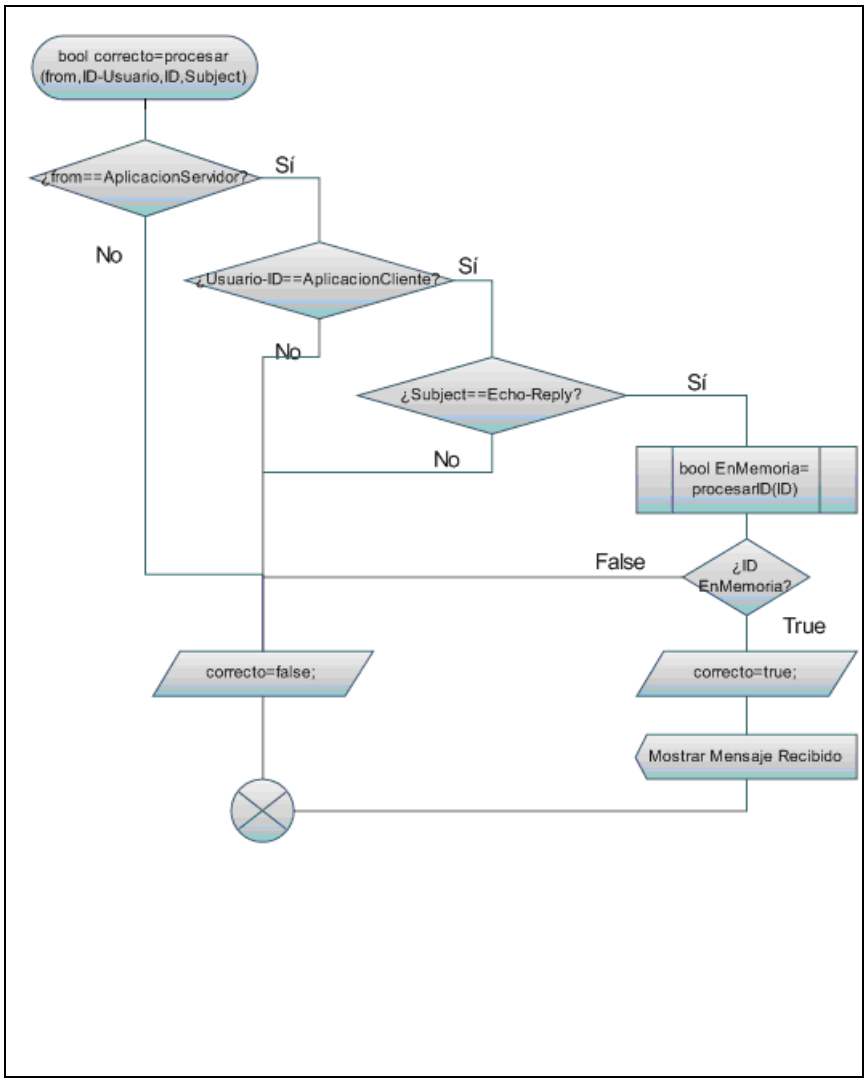


Figura 25: Diagrama de flujo del método procesar de la Subcapa de Procesamiento Receptora de la aplicación *Cliente.*

3.4.4.2 Capa de Aplicación Cliente.

Es la última capa del diseño propuesto, encargada de mostrar una interfaz al usuario que permite la introducción y visualización de los diferentes parámetros generados por el sistema.

En el caso del cliente, se muestra una pantalla donde el usuario introduce el texto que desea enviar. Además, para la comprobación del correcto funcionamiento de la aplicación, se han incluido dos pantallas del tipo *ListBox()* que muestran los mensajes enviados y los mensajes recibidos, lo que permite comprobar fácilmente la correspondencia entre los mensajes enviados y los recibidos.

Por último, se muestra una ventana de *Log*, donde se refleja el intercambio de mensajes y comandos entre la aplicación *Cliente* y el servidor de correo de *Gmail*.

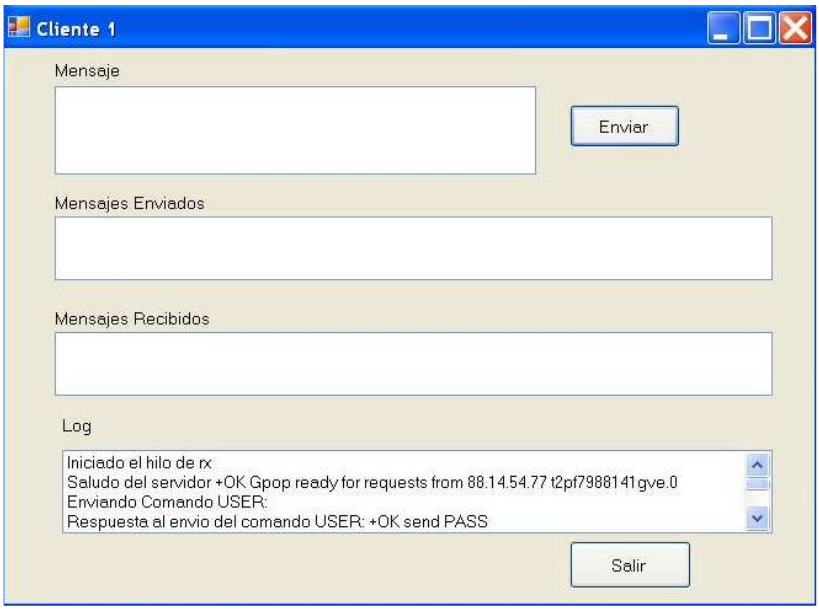


Figura 26: Capa de aplicación de la aplicación *Cliente.*

Una vez explicadas todas las capas de la aplicación *Cliente*, se muestra un diagrama UML de Secuencia que muestra todas las páginas que conforman la aplicación, así como el envío y recepción de mensajes entre ellas, y su interacción con el usuario.

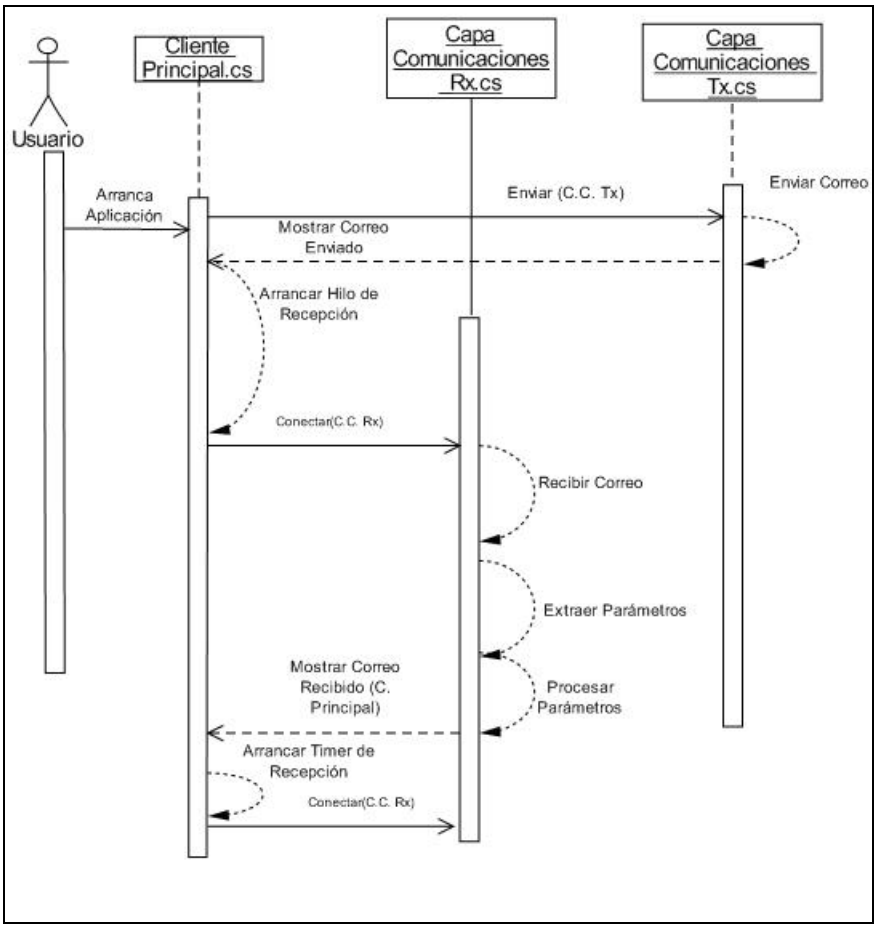


Figura 27: Diagrama UML de Secuencias de la Aplicación *Cliente*.

3.4.2 Aplicación Servidor.

La aplicación *Servidor* se encarga de recibir los correos presentes en el servidor de corrreo de *Gmail*, para pasar a procesarlos y enviar su respuesta de nuevo a *Gmail* para que los clientes puedan acceder a ellos y verificar la respuesta a los mensajes que enviaron.

Al igual que en el caso de las aplicaciones *Cliente*, para chequear la llegada de nuevos correos en el servidor, se ha implementado un *Timer* que realiza dicha comprobación cada cierto tiempo.

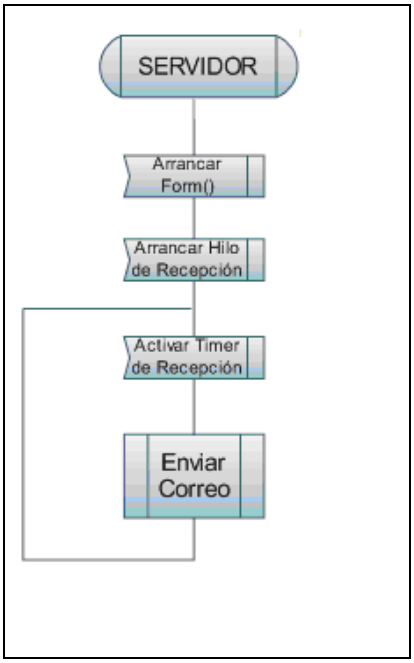


Figura 28: Diagrama de estados principal de la aplicación *Servidor*.

A continuación, pasamos a describir las diferentes capas que componen la aplicación *Servidor*.

3.4.2.1 Capa de Comunicaciones Servidor.

Como se comentó, esta capa es común para tanto para la aplicación *Cliente* como *Servidor*, estando subdividida en una Subcapa de Comunicaciones Transmisora, para el envió de correos al servidor de *Gmail*, y una Subcapa de Comunicaciones Receptora, encargada de la recepción de los correos procedentes de *Gmail*, tal y como se puede observar el la figura 20.

3.4.2.1.1 Subcapa de Comunicaciones Transmisora.

Esta subcapa es idéntica a la descrita para el caso de la aplicación *Cliente*, sólo que en este caso, no es necesario llamar a la clase *RandomID* para generar un identificador de mensaje, ya que éste es extraído por la Capa de Procesamiento del mensaje recibido y asignado de nuevo a la cabecera *"ID:"* para que el cliente relacione este correo con el que envió anteriormente.

Al igual que el identificador de mensaje, se fija una cabecera con el nombre de usuario del cliente que envió el correo para que, de nuevo el cliente pueda relacionar el mensaje enviado con su respuesta.

También señalar que en este caso el fichero *Parámetros_Configurables.xml* es diferente al caso anterior, ya que la aplicación que envía el correo es diferente, *aplicacionservidor@Gmail.com.*

Por lo demás, esta subcapa es la encargada de recibir el correo electrónico ya formado proveniente de la Capa de Procesamiento, para su posterior envío al servidor de correo de *Gmail*. Para el envío del correo electrónico se ha utilizado la clase *SmtpClient*

Todo lo descrito con anterioridad se muestra en el siguiente flujograma para una mejor comprensión:

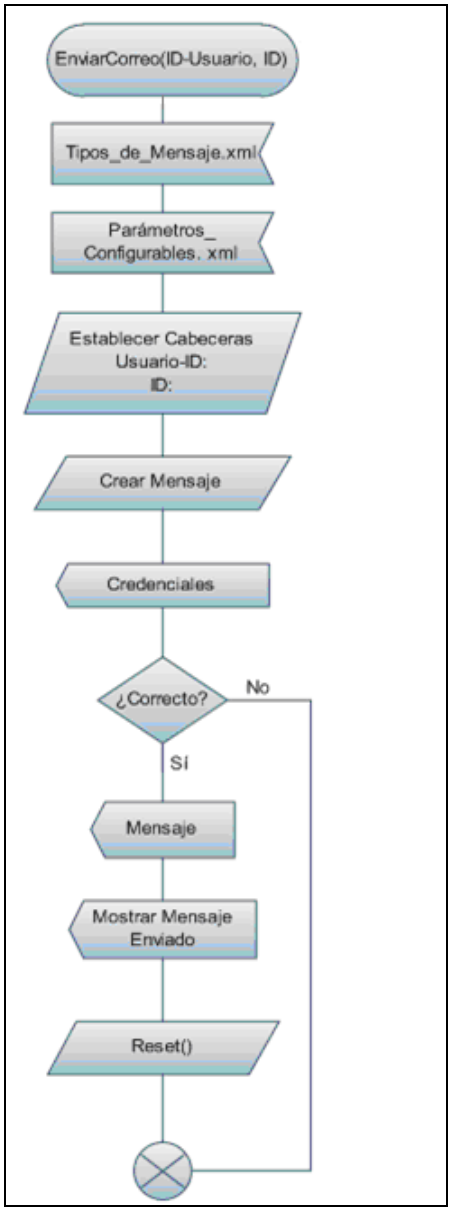


Figura 29: Flujograma de la Subcapa de Comunicaciones Receptora de la aplicación *Servidor.*

3.4.2.1.2 Subcapa de Comunicaciones Receptora.

Capa idéntica a la Subcapa de Comunicaciones Receptora de la aplicación *Cliente*, donde se produce el intercambio de mensajes entre el servidor de correo de *Gmail* y nuestra aplicación *Servidor*.

El siguiente flujograma muestra todo el proceso de envío de comandos:

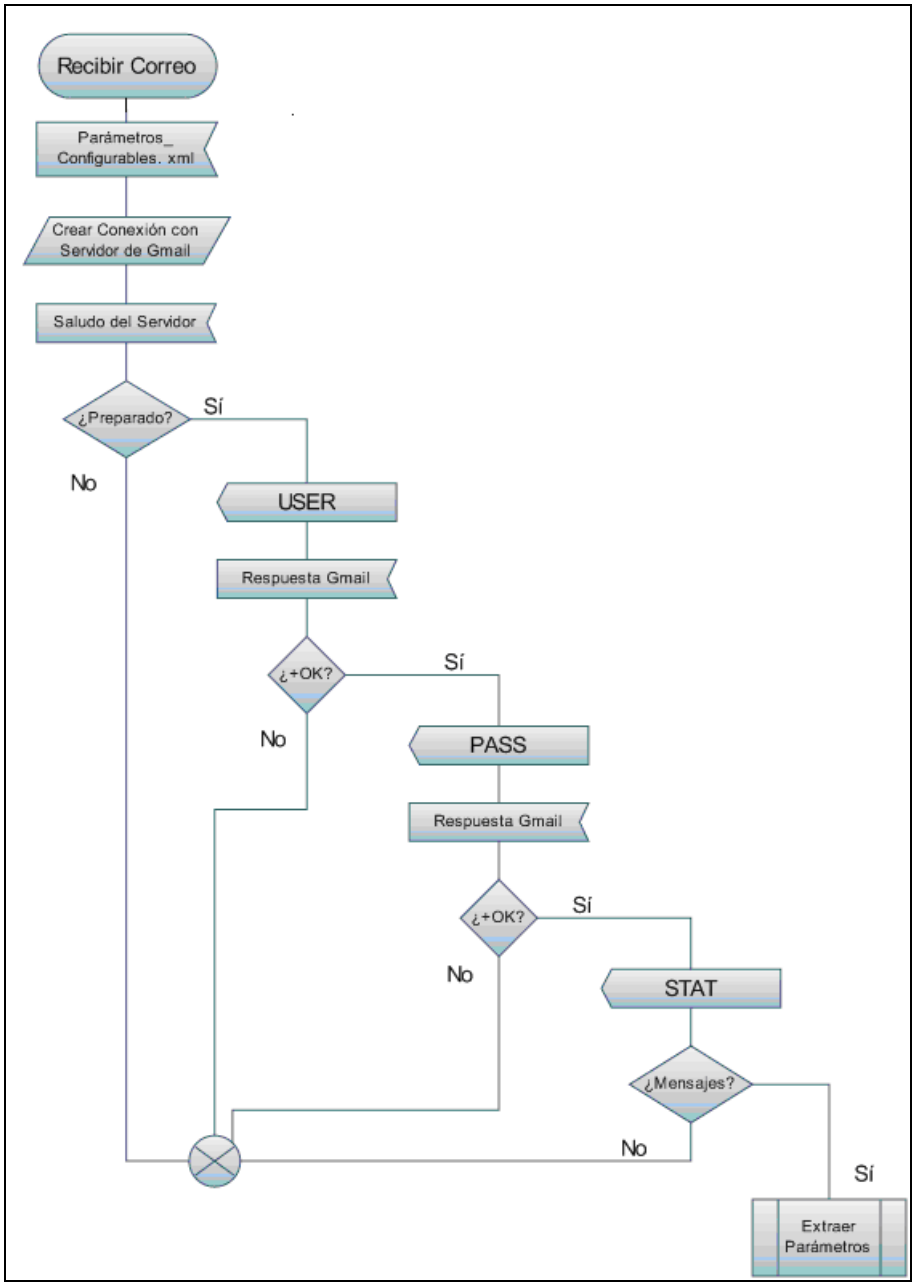


Figura 30: Diagrama de flujo de la Subcapa de Comunicaciones Receptora de la aplicación *Servidor*.

3.4.2.2 Capa de Procesamiento Servidor.

Al igual que en el caso anterior, esta capa estará dividida en dos subcapas, denominadas Subcapa de Procesamiento Transmisora, encargada de fijar todos los parámetros necesarios para la composición del correo electrónico a enviar; y Subcapa de Procesamiento Receptora, cuya función es procesar los mensajes de correo electrónico en función de los parámetros buscados.

Para su explicación se ha alterado el orden llevado en el caso de la aplicación *Cliente*, para una mejor comprensión, ya que primero recibimos el correo, lo procesamos y enviamos la respuesta.

3.4.2.2.1 Subcapa de Procesamiento Receptora.

Una vez comprobada la existencia de mensajes en el servidor de correo de *Gmail*, esta subcapa se encargará de su descarga y procesamiento.

En primer lugar, para cada uno de los mensajes presentes en el servidor, se envía el comando *"RETR"*, que nos devuelve el mensaje completo, incluidas las cabeceras, en una cadena de texto.

A continuación, se llama al método *ExtraerParámetros()*, donde al igual que el caso anterior, de la cadena devuelta al enviar el comando *"RETR"* se extraen diversas cabeceras, en este caso sólo nos harán falta *"ID-Usuario:","ID:"* y *"Subject:"*.

De nuevo se llama al método *Procesar()* que realiza las correspondientes operaciones en función del tipo de mensaje; si estas operaciones se han llevado a cabo de forma satisfactoria se pasa al borrado del mensaje recibido mediante el envío de los comandos *"DELE"* y *"QUIT"*, y se envían los parámetros necesarios a la Subcapa de Comunicaciones Transmisora para que forme y envíe el correo de respuesta al servidor de *Gmail*.

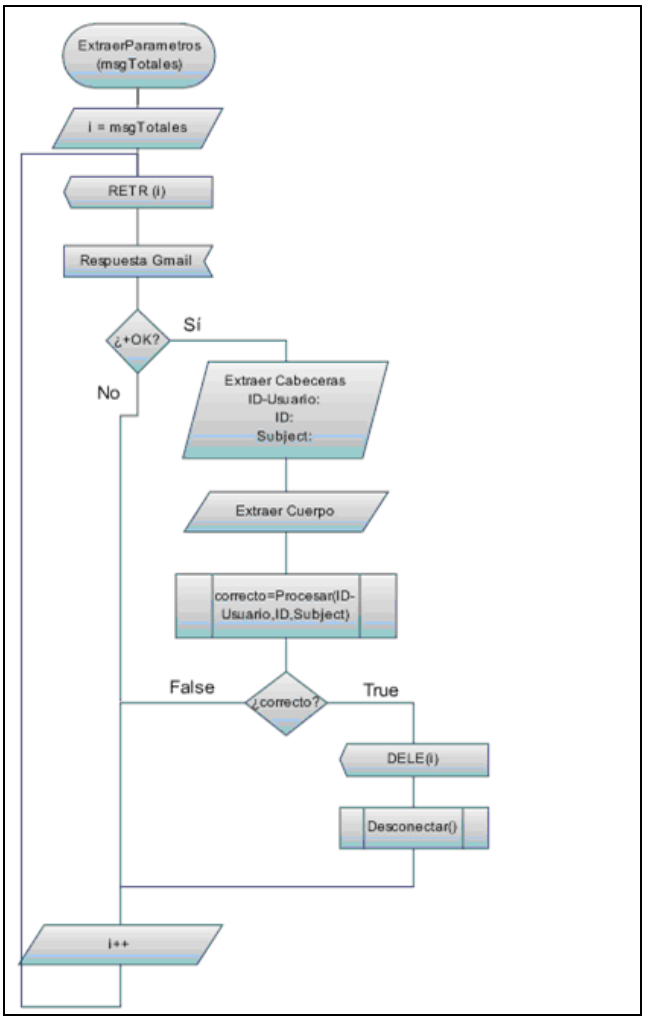


Figura 31: Flujograma de la Subcapa de Procesamiento Receptora de la aplicación *Servidor*.

En último lugar, se describe el método *Procesar()* al cual se le pasa como argumento los parámetros de las cabeceras extraídos anteriormente, devolviendo como resultado una variable booleana que indica si el correo electrónico es el esperado o no. Las comprobaciones que se realizan son las siguientes:

1º) Comprobamos, mediante el parámetro extraído de la cabecera *"ID-Usuario:"*, si el mensaje recibido no proviene de la aplicación "Servidor", así nos aseguramos que proviene de un cliente y no recogemos los mensajes de respuesta enviados por nuestra aplicación anteriormente.

2º) Mediante el asunto del correo, suministrado por la cabecera *"Subject:"*, comprobamos que el asunto recibido se corresponde con el asunto enviado. En nuestro caso, ante la llegada de un mensaje con asunto *EchoRequest*, fijamos el asunto del correo de respuesta como *EchoReply*.

Si todas las comprobaciones anteriores son correctas pasamos a enviar el correo.

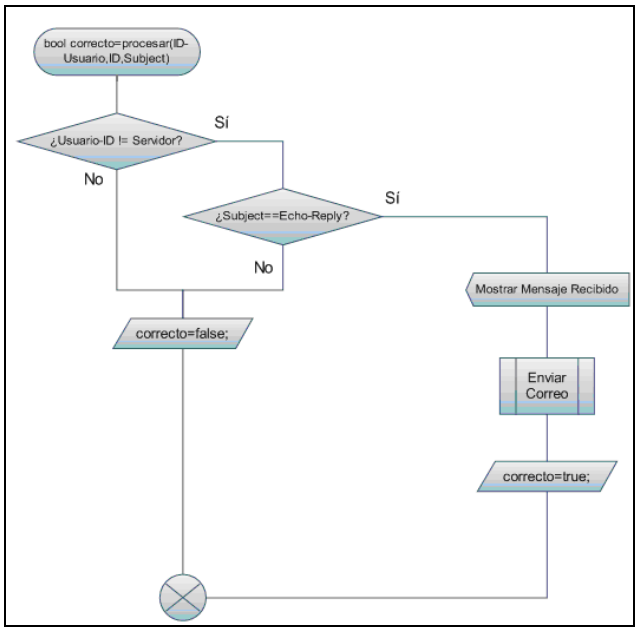


Figura 32: Flujograma del método *Procesar()* de la Subcapa de Procesamiento de la aplicación *Servidor*.

3.4.2.2.2 Subcapa de Procesamiento Transmisora.

Esta subcapa es la encargada de proporcionar todos los datos necesarios para enviar un correo electrónico: dirección origen, dirección destino, asunto, cuerpo del mensaje, archivos adjuntos, etc. Todos estos datos son proporcionados por la Subcapa de Procesamiento Receptora que, como hemos visto anteriormente es la encargada de extraer todos estos parámetros del correo electrónico recibido. Su diagrama de flujo es idéntico al de la aplicación *Cliente.*

3.4.2.3 Capa de Aplicación Servidor.

Esta capa, la más alta en la arquitectura de comunicaciones implementada, al igual que en el caso de la aplicación *Cliente*, está formada por una interfaz gráfica que permite, de forma intuitiva observar el envío, descarga e intercambio de mensajes entre el servidor de correo de *Gmail* y nuestra aplicación *Servidor*.

Esta interfaz gráfica consta de tres *ListBox*. El primero de ellos se encarga de mostrar por pantalla los mensajes recibidos por la aplicación “Servidor” y que van destinados a ella. La segunda ventana muestra los mensajes una vez procesados y que son enviados al servidor de *Gmail*; por último, el tercer *ListBox* muestra el intercambio de mensajes que llevan a cabo ambas aplicaciones.

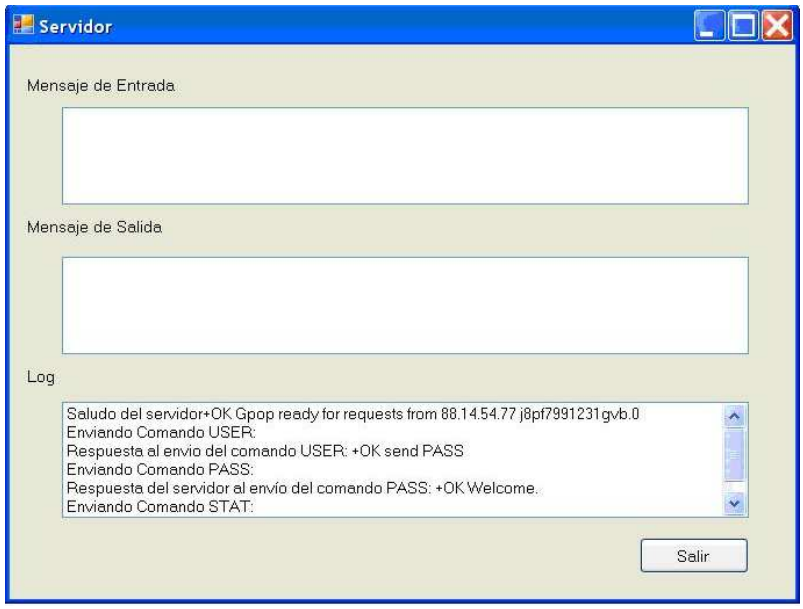


Figura 33: Capa de Aplicación de la aplicación *Servidor*.

Para concluir se muestra el diagrama UML de Secuencia de la aplicación *Servidor*:

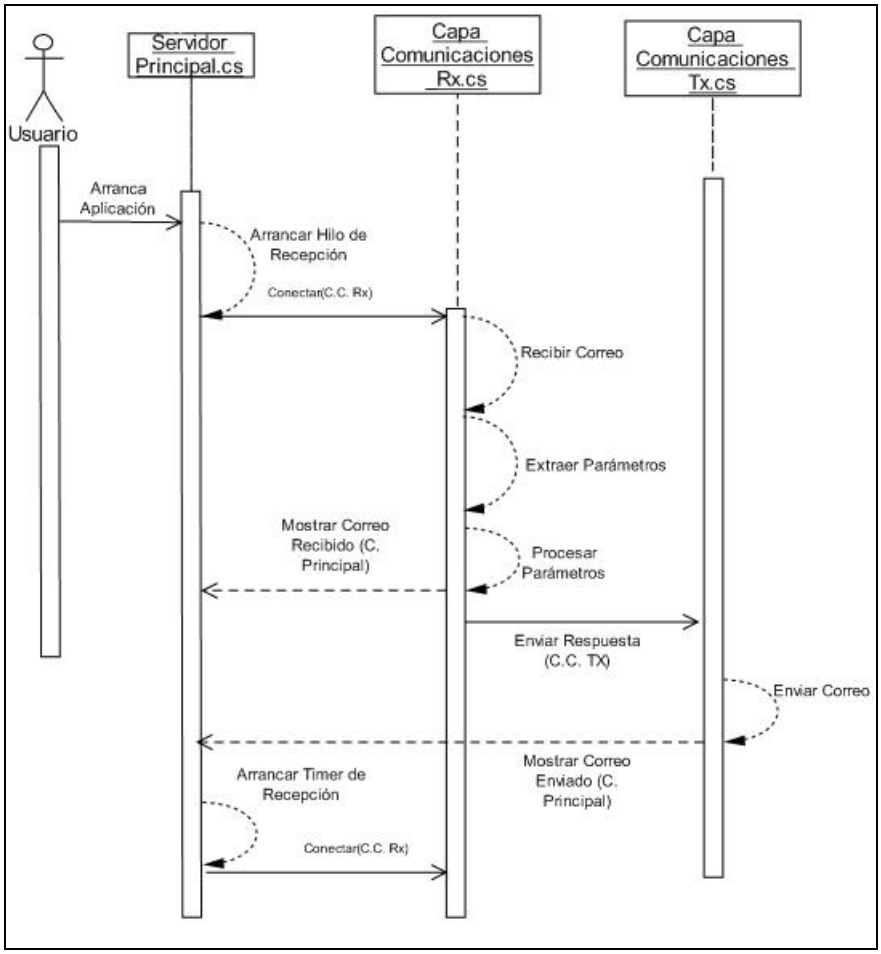


Figura 34: Diagrama UML de Secuencias de la aplicación *Servidor*.

Capítulo 4: CONFIGURACIÓN DE LAS APLICACIONES

4. Configuración de las aplicaciones.

En los siguientes apartados se va a proceder a explicar detalladamente los pasos a seguir para configurar las diferentes aplicaciones, a fin de que el usuario final de dichas aplicaciones pueda adaptarlas a sus necesidades.

4.1 Configuración de la Aplicación Cliente.

En primer lugar señalar que, como se ha comentado durante todo este trabajo, esta arquitectura está diseñada para funcionar con varias aplicaciones *Cliente*, por lo tanto, a la hora de crear nuevos *Clientes*, simplemente hay que realizar una copia de los ficheros que contienen todas las clases de la implementación, y adaptar los archivos XML a la aplicación que se quiere crear.

En el caso de la aplicación Cliente deberemos acceder en primer lugar al archivo *Parámetros_Configurables.xml* y configurar cada uno de los nodos que definen las propiedades de cada protocolo.

```
<?xml version="1.0" encoding="utf-8" ?>
<protocolos>
	<smtp>
		<servidor>smtp.Gmail.com</servidor>
		<origen>aplicacioncliente1@Gmail.com</origen>
		<destino>aplicacionservidor@Gmail.com</destino>
		<puerto>587</puerto>
		<contrasena>upct2007</contrasena>
	</smtp>
	<pop3>
		<servidor>pop.Gmail.com</servidor>
		<puerto>995</puerto>
		<usuario>aplicacioncliente1@Gmail.com</usuario>
		<contrasena>upct2007</contrasena>
		<aplicacionservidor>aplicacionservidor@Gmail.com</aplicacionservidor>
	</pop3>
</protocolos>
```

Figura 35: Archivo “Parámetros Configurables” de la Aplicación *Cliente.*

Para el caso del protocolo SMTP, tenemos los siguientes nodos:

- `<servidor>:` define la dirección del servidor SMTP que proporciona nuestro proveedor de correo.
- `<puerto>:` indica el puerto del servidor SMTP donde se presta este servicio.
- `<origen>:` indica la dirección de correo de la cuenta del cliente creado; este es el parámetro que deberemos cambiar al crear un nuevo cliente.
- `<contrasena>:` contraseña de la cuenta de correo con la que se va a acceder al servidor SMTP para enviar el mensaje.
- `<destino>:` indica la dirección de correo de la aplicación *Servidor* encargada de procesar los correos electrónicos.

En el caso del protocolo POP3, tenemos los siguientes nodos:

- `<servidor>:` dirección del servidor POP3 al cual se accederá para descargar los mensajes de correo electrónico.

- `<puerto>:` número de puerto donde se presta el servicio POP3.
- `<usuario>:` indica la dirección de correo de la cuenta del cliente creado que permite acceder al servidor POP3; este es el parámetro que deberemos cambiar al crear un nuevo cliente.
- `<contrasena>:` contraseña de la cuenta de correo con la que se va a acceder al servidor POP3 para descargar los mensajes.
- `<aplicacionservidor>:` Indica la procedencia de los mensajes descargados, sirve para verificar que los mensajes que llegan al servidor POP3 proceden de la aplicación *Servidor* correcta.

En la siguiente tabla se muestran los valores de configuración para el caso de utilizar cuentas de correo de *Gmail*.

Servidor de Correo Entrante (POP3) **Nota: Requiere SSL**	**Servidor: pop.*Gmail*.com** **Usar SSL: Sí** **Puerto: 995**
El servidor de correo saliente (SMTP) **Nota: Requiere TLS**	Servidor: smtp.*Gmail*.com Usar autenticación: Sí Usar STARTTLS: Sí (Nota: En algunos clientes se denomina SSL) Puerto: 465 ó 587
Nombre de cuenta	Dirección de usuario de *Gmail* (incluido *@Gmail.com*)
Dirección de correo electrónico	Dirección de correo electrónico completa de *Gmail* (*nombredeusuario@Gmail.com*)
Contraseña	Contraseña de *Gmail*.

Tabla 24: Parámetros de configuración del cliente de correo “aplicación Cliente”.

En segundo lugar, deberemos acceder al archivo *Tipos_de_Mensaje.xml*, donde especificaremos los mensajes a enviar por la aplicación cliente, el tipo de mensaje, el asunto que llevará, el cuerpo… Como se ha explicado anteriormente, al utilizar archivos XML, esta aplicación es fácilmente ampliable para todo tipo de necesidades, simplemente añadiendo un nuevo nodo `<mensaje>`.

```
<?xml version="1.0" encoding="utf-8" ?>
<mensajes>
      <mensaje>
            <tipo>ECHO</tipo>
            <asunto_enviado>Echo Request</asunto_enviado>
            <asunto_esperado>Echo Reply<asunto_esperado>
            <texto>Echo Request del mensaje</texto>
      </mensaje>
</mensajes>
```

Figura 36: Archivo “Tipos de Mensaje” de la Aplicación *Cliente.*

- `<tipo>:` identificador que señala el tipo de mensaje a tratar; este será el nodo que deberemos analizar, para determinar la funcionalidad de la aplicación.
- `<asunto_enviado>:` indica el asunto que se insertará en el mensaje.
- `<asunto_esperado>:` indica el asunto esperado al recibir un mensaje de correo electrónico de un tipo determinado.
- `<texto>:` indica el cuerpo del mensaje a enviar; puede ser sustituido por datos procedentes del exterior.

4.2 Configuración de la Aplicación Servidor.

La configuración de la aplicación *Servidor* es similar a la realizada para el caso de la aplicación *Cliente*, sustituyendo en cada caso los nombres de usuario y contraseña por los de la aplicación que va a actuar como servidor en el archivo *Parámetros_Configurables.xml.*

```
<?xml version="1.0" encoding="utf-8" ?>
<protocolos>
      <smtp>
            <servidor>smtp.Gmail.com</servidor>
            <origen>aplicacionservidor@Gmail.com</origen>
            <puerto>587</puerto>
            <contrasena>upct2007</contrasena>
      </smtp>
      <pop3>
            <servidor>pop.Gmail.com</servidor>
            <puerto>995</puerto>
            <usuario>aplicacionservidor@Gmail.com</usuario>
            <contrasena>upct2007</contrasena>
      </pop3>
</protocolos>
```

Figura 37: Archivo "Parámetros Configurables" de la Aplicación *Servidor.*

Para el caso del archivo *Tipos_de_Mensaje.xml* deberemos adaptar la creación de nuevos mensajes en función de la aplicación a considerar, es decir, habrá una interrelación entre los tipos de mensaje enviados por la aplicación *Cliente* y los recibidos por la aplicación "Servidor".

```
<?xml version="1.0" encoding="utf-8" ?>
<mensajes>
      <mensaje>
            <tipo>ECHO</tipo>
            <asunto_esperado>Echo Request</asunto_esperado>
            <asunto_respuesta>Echo Reply</asunto_respuesta>
            <texto>Echo Reply del mensaje recibido</texto>
      </mensaje>
</mensajes>
```

Figura 38: Archivo "Tipos de Mensaje" de la Aplicación *Servidor.*

4.3 Configuración de la cuenta de correo de *Gmail* que actúa como servidor intermedio.

Como se ha comentado anteriormente, existe una versión híbrida entre el correo *web* y los clientes de correo. *Gmail*, a pesar de ser un cliente *web*, permite habilitar el acceso POP, para la descarga de mensajes. A continuación se muestran una serie de pasos para la configuración del acceso POP.

En primer lugar, debemos habilitar el acceso POP en la cuenta de *Gmail* que actúa como servidor de correo (en este proyecto es *aplicacionservidor@Gmail.com*). Para ello, se siguen una serie de pasos que se muestran a continuación:

- Acceder a la cuenta de *Gmail* (*www.Gmail.com*).

Figura 39: Acceso a la cuenta de *Gmail*.

- Una vez dentro, hacer clic en **Configuración,** en la parte superior de cualquier página de *Gmail*.

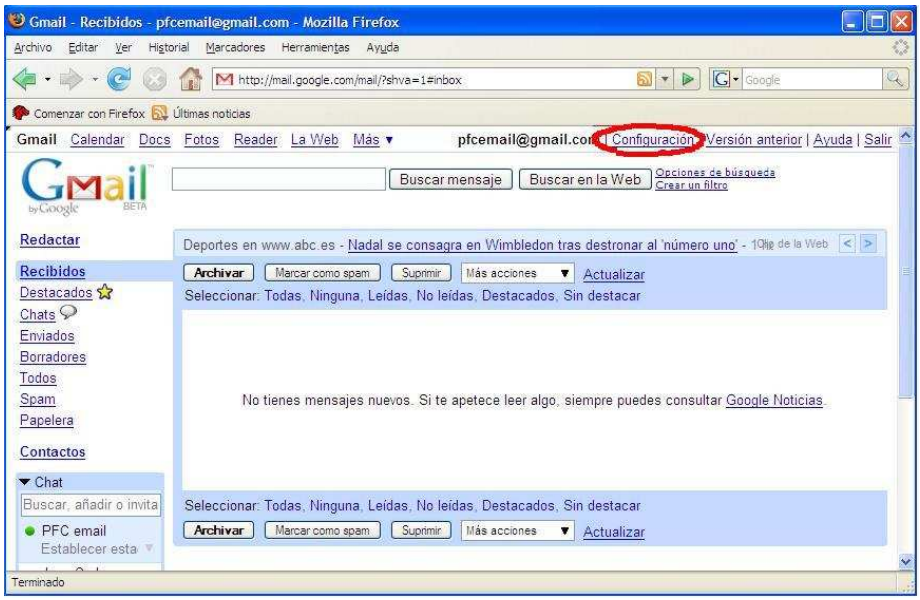


Figura 40: Acceso al menú de Configuración de *Gmail*.

- Hacer clic en **Reenvío y correo POP**, situado en el cuadro naranja **Configuración.**(1)

- Seleccionar **Habilitar POP para todos los mensajes** o **Habilitar POP para los mensajes que se reciban a partir de ahora**.(2)

- Elegir la acción que deseas que realicen tus mensajes de *Gmail* una vez que hayas accedido a ellos a través del protocolo POP. En este proyecto se ha optado por eliminar los mensajes del servidor una vez descargados y procesados.(3)

- Por último configurar el cliente POP y hacer clic en **Guardar cambios.**

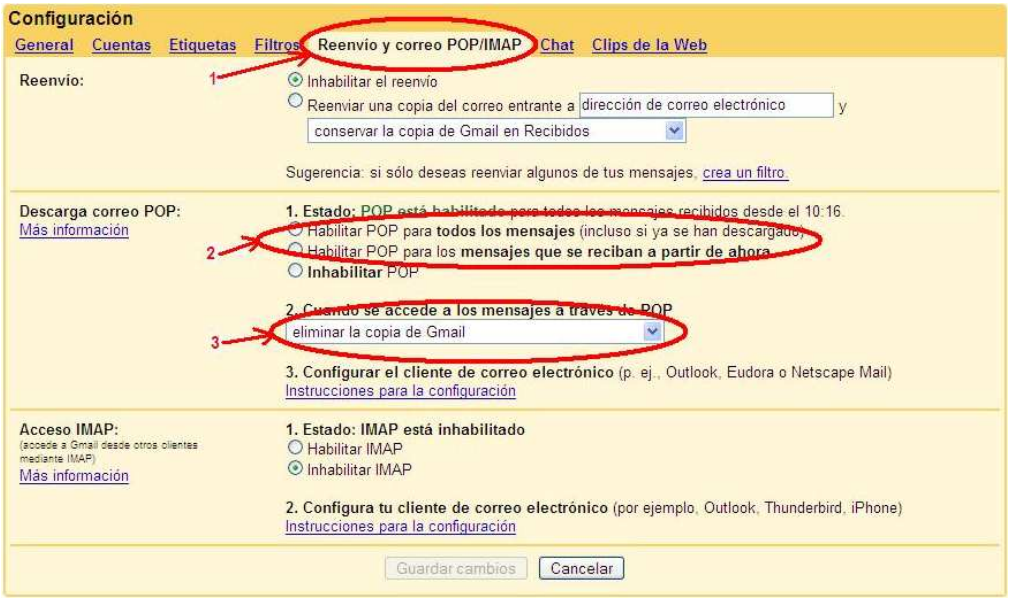


Figura 41: Configuración del acceso POP de *Gmail*.

Capítulo 5: EJEMPLO DE FUNCIONAMIENTO DE LAS APLICACIONES

5. Ejemplo de funcionamiento de la aplicación

En este punto se va a presentar un sencillo ejemplo del funcionamiento de las aplicaciones implementadas, donde se podrá comprobar de una manera fácil e intuitiva el funcionamiento de este proyecto.

Los pasos a seguir son los siguientes:

1. Arrancar el formulario de envío del cliente

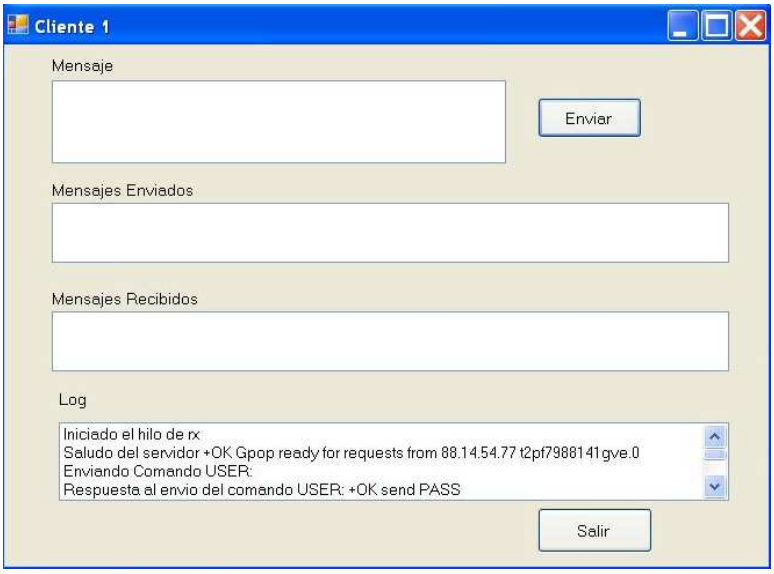


Figura 42: Interfaz gráfica de la aplicación *Cliente*.

2. Escribir el mensaje que se desea enviar, pulsando a continuación el botón de envío; se puede observar cómo aparece el mensaje enviado en la pantalla de Mensajes Enviados, con su correspondiente ID (en este caso el ID generado es JHTWYAYOHTNKNNT) y asunto *"Echo Request":*

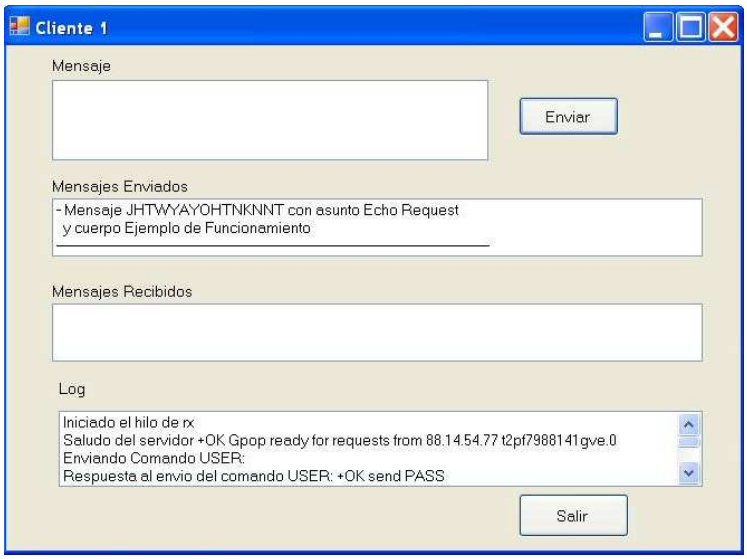


Figura 43: Vista del mensaje a procesar en la aplicación *Cliente*.

3. El mensaje enviado por la aplicación *Cliente* llega a la cuenta de correo de la aplicación *Servidor*:

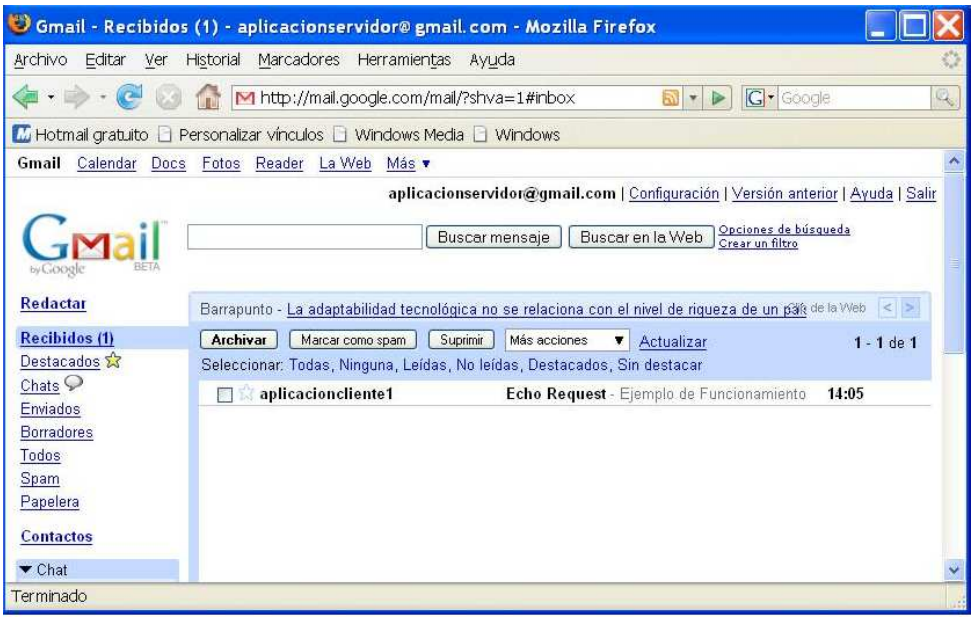


Figura 44: Llegada al servidor de correo de *Gmail* del mensaje sin procesar proveniente de la aplicación *Cliente*.

4. La aplicación *Servidor* se descarga el mensaje recibido, mostrándolo por la ventana de Mensajes de Entrada, para proceder a su procesado.

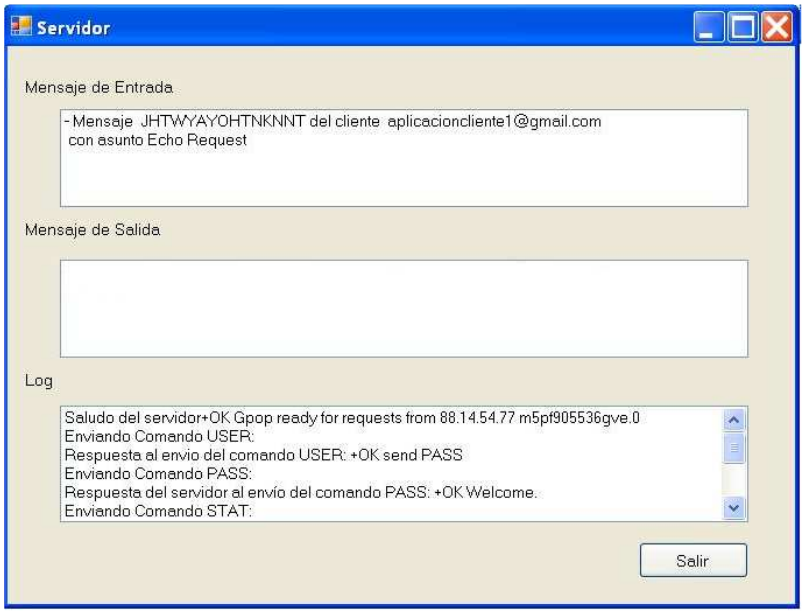


Figura 45: Recepción en la aplicación *Servidor* del mensaje sin procesar.

5. Una vez procesado, el mensaje con asunto *Echo Reply* es enviado de nuevo al servidor de correo de *Gmail*, mostrándose el correo ya analizado en la ventana de Mensajes de Salida.

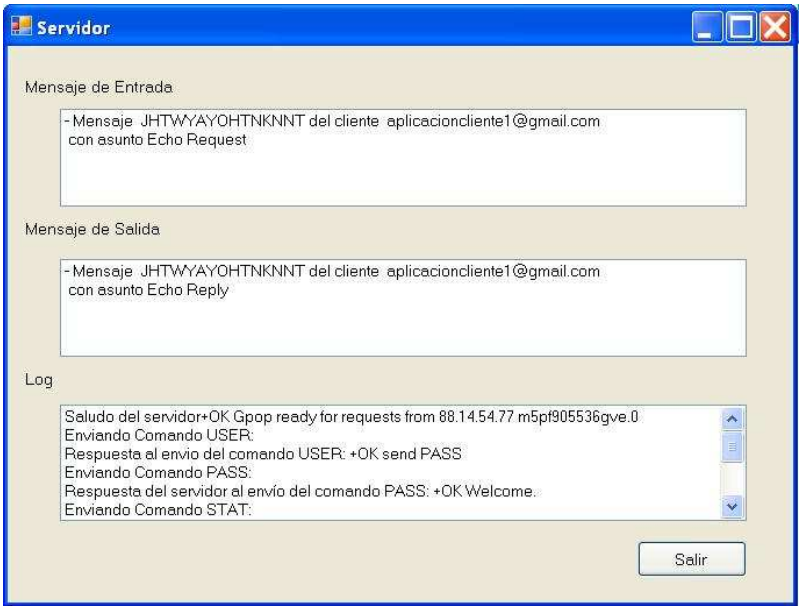


Figura 46: Vista del mensaje procesado en la Aplicación *Servidor*.

6. El mensaje de respuesta llega a la cuenta de correo de la aplicación *Cliente.*

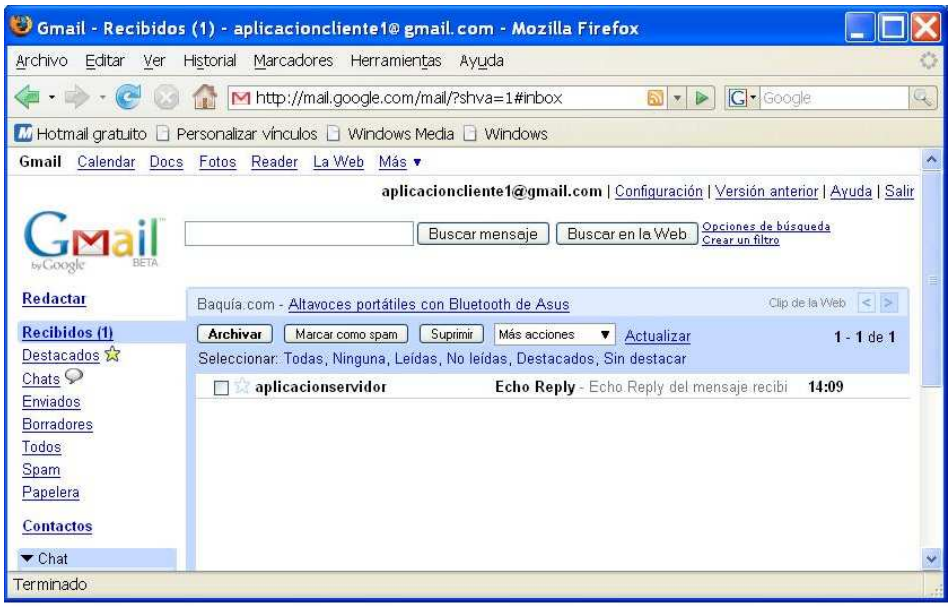


Figura 47: Llegada del mensaje procesado a la cuenta de correo de la aplicación *Cliente*.

7. La aplicación *Cliente* descarga dicho mensaje y, tras realizar las verificaciones necesarias para comprobar que es el mensaje que esperaba, lo muestra por la Ventana de Mensajes Recibidos.

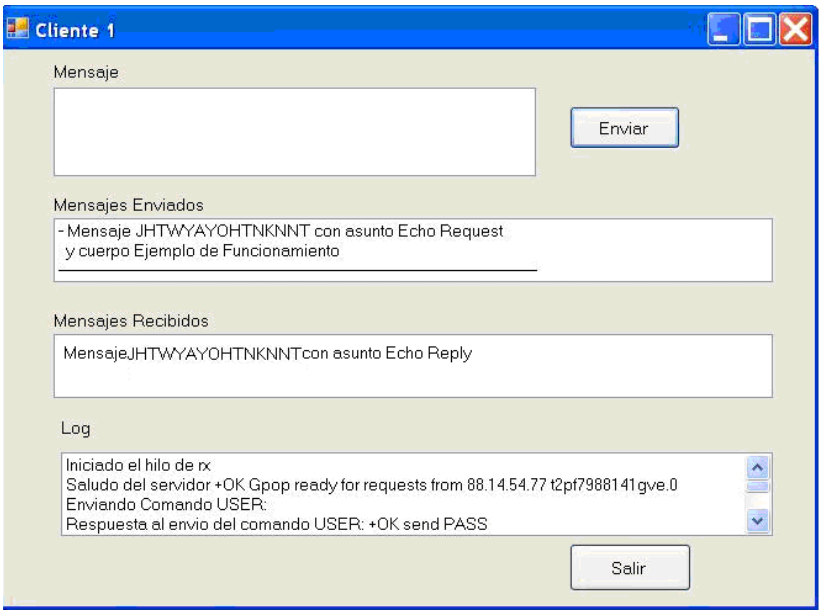


Figura 48: Recepción en la aplicación *Cliente* del mensaje procesado.

Capítulo 6: PRUEBAS REALIZADAS YCONCLUSIONES

6. Pruebas realizadas y conclusiones extraídas.

6.1 Pruebas realizadas

La última etapa, una vez finalizado el desarrollo de las aplicaciones, ha consistido en la depuración de las mismas para comprobar la robustez y el correcto funcionamiento de éstas.

Antes de pasar a exponer los resultados obtenidos, conviene mencionar las condiciones bajo las cuales se llevaron a cabo estas pruebas:

- El equipo empleado fue un Pentium IV a 1.5GHz.
- Sistema operativo Microsoft Windows XP.
- Conexión a Internet de banda ancha ADSL con una capacidad de 3Mbps.

Con dichas pruebas, además de comprobar el correcto funcionamiento de las aplicaciones implementadas, se han hallado los diversos retardos implicados en las comunicaciones entre las aplicaciones *Cliente* y la aplicación *Servidor*, la influencia que tienen los distintos *Timers* que intervienen en dichas aplicaciones y en qué medida afecta el aumento del número de clientes al retardo del sistema. Para ello, se han simulado una serie de escenarios donde se ha variado la tasa de envío de mensajes de los *Clientes*, dándose dos situaciones posibles con el envío aleatorio de mensajes en periodos de 30sg y de 1 minuto. Por otro lado, se ha variado el tiempo del *Timer* de chequeo de la aplicación *Servidor* entre los valores de 15sg y 30sg. El *Timer* de chequeo de las aplicaciones *Clientes* se ha mantenido constante, a excepción de un escenario donde este tiempo se ha decrementado para paliar los efectos de una simulación donde los resultados no han sido los esperados.

Por último, comentar que cada aplicación *Cliente* ha enviado un total de 50 mensajes, y que los tiempos de transferencia se han obtenido a partir del asunto de cada correo electrónico que contenía la hora en la cual se enviaron y recibieron los correos electrónicos.

A continuación se muestran los escenarios implementados:

6.1.1 Simulaciones con 1 aplicación Cliente

6.1.1.1 Escenario 1

- Condiciones:
    - Tasa de envío de la aplicación *Cliente* aleatoria en un intervalo de 30sg.
    - *Timer* de chequeo de la aplicación *Servidor* de 15sg.
    - *Timer* de chequeo de la aplicación *Cliente* de 15sg.

- Resultados de la simulación:

    - Para este primer caso, obtenemos un tiempo medio de recepción de los mensajes por parte de los *Clientes* de 23.34sg, que como vemos se ajusta perfectamente al valor de los *Timers* de ambas aplicaciones, ya que el sistema se encuentra descargado de mensajes y la velocidad de procesamiento es mayor.

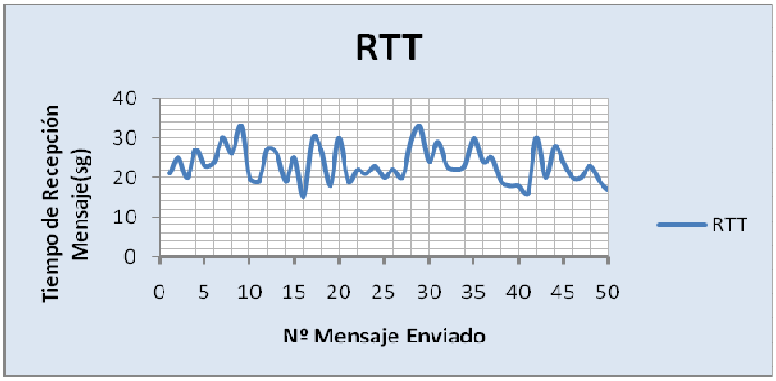


Figura 49: RTT del escenario 1 de la simulación.

6.1.1.2 Escenario 2

- Condiciones:
    - Tasa de envío de la aplicación *Cliente* aleatoria en un intervalo de 30sg.
    - *Timer* de chequeo de recepción de correo en la aplicación *Servidor* de 30sg.
    - *Timer* de chequeo de recepción de correo en la aplicación *Cliente* de 15sg.

- Resultados de la simulación:

    - En este caso lo que se ha variado ha sido el *Timer* de chequeo de la aplicación *Servidor*, manteniéndose constantes el resto de parámetros. Como se puede observar en la gráfica inferior, el tiempo medio de recepción de los mensajes en el *Cliente* ha aumentado, alcanzando un valor de 57.92sg. Esto es lógico, ya que ahora el *Servidor* tarda más tiempo en chequear la llegada de correos y, por tanto, el procesamiento se ralentiza, sobretodo al final de la simulación cuando el número de correos que hay presentes en el buzón del *Servidor* es mayor.

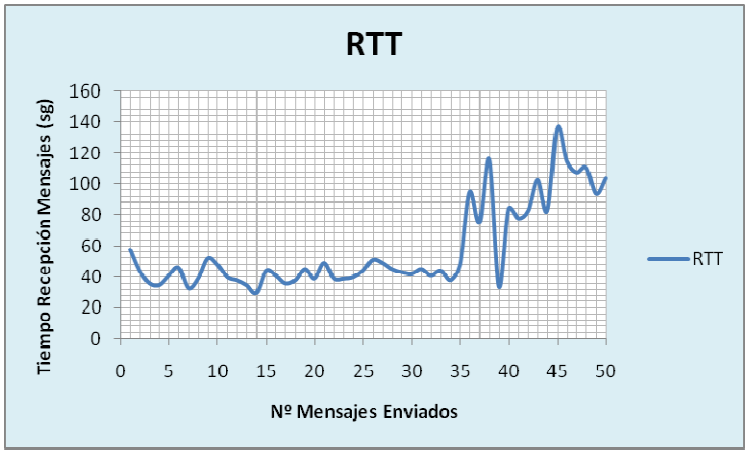


Figura 50: RTT del Escenario 2 de la simulación.

6.1.1.3 Escenario 3

- Condiciones:
  - Tasa de envío de la aplicación *Cliente* aleatoria en un intervalo de 60sg.
  - *Timer* de chequeo de recepción de correo en la aplicación *Servidor* de 15sg.
  - *Timer* de chequeo de recepción de correo en la aplicación *Cliente* de 15sg.

- Resultados de la simulación:

  - En este tercer escenario, se ha duplicado el periodo en el que el *Cliente* envía mensajes al *Servidor*. El tiempo medio de recepción de los mensajes es de 35.36sg que, contrariamente a lo esperado, es superior a los tiempos obtenidos en el escenario 1 donde la tasa de envío era mayor.

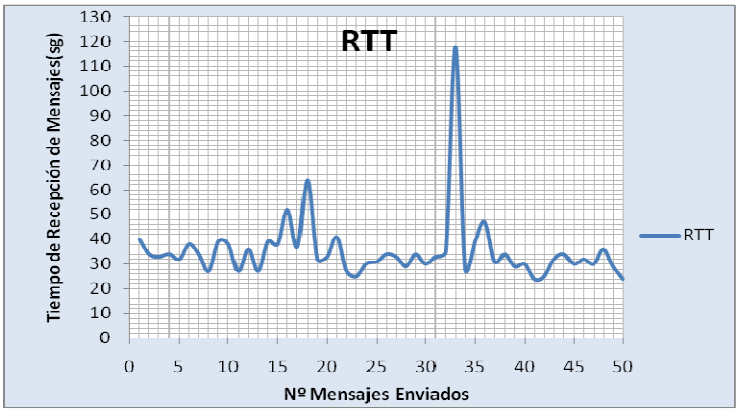


Figura 51: RTT del Escenario 3 de la simulación.

6.1.1.4 Escenario 4

- Condiciones:
  - Tasa de envío de la aplicación *Cliente* aleatoria en un intervalo de 60sg.
  - *Timer* de chequeo de la aplicación *Servidor* de 30sg.
  - *Timer* de chequeo de la aplicación *Cliente* de 15sg.

- Resultados de la simulación:

  - Tiempo medio de recepción en el cliente de los mensajes de respuesta de 38.94sg. Al igual que en el caso de la simulación 2, este tiempo es superior al caso en el que el *Timer* de chequeo del *Servidor* es de 15sg; sin embargo, es este caso dicha diferencia es menor debido a que el intervalo en el que se generan mensajes por parte del *Cliente* es mayor, con lo que el buzón del *Servidor* se encuentra con menos mensajes y su velocidad de procesamiento es mayor.

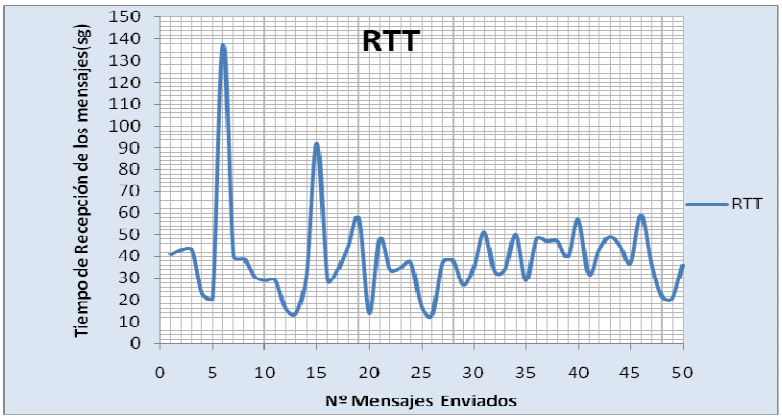


Figura 52: RTT del Escenario 4 de la simulación.

6.1.1.5 Comparativa de los 4 escenarios implementados

- En primer lugar se ha fijado como intervalo de transmisión de los mensajes por parte de los *Clientes* un periodo de 30sg, y lo que se ha variado ha sido el *Timer* de chequeo de la aplicación *Servidor*. Se ha obtenido la siguiente gráfica:

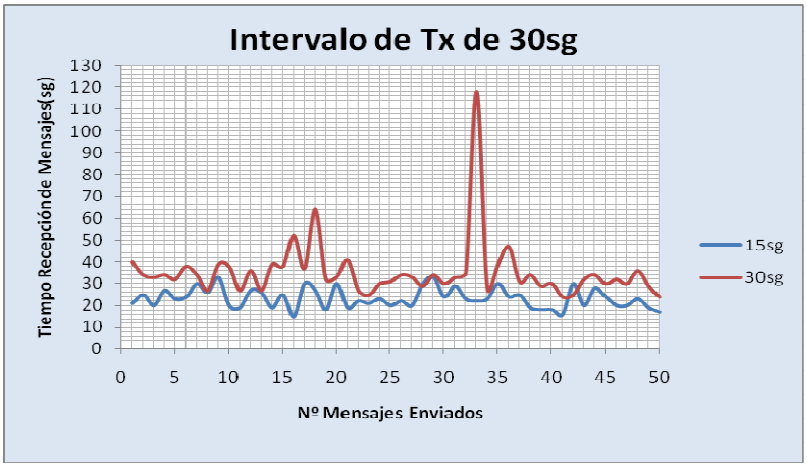


Figura 53: Comparativa Escenarios 1-4 de la simulación (Intervalo de Tx de 30sg)

- Se observa que en el caso del *Timer* de 30sg el tiempo de recepción de los mensajes es ligeramente mayor que en el caso de tener un *Timer* de 15sg, por lo explicado anteriormente de que el *Servidor* comprueba la llegada de nuevos correos en un periodo de tiempo mayor, con lo que los mensajes de respuesta se retardan en su llegada al *Cliente*.

- En segundo lugar, se ha modificado el intervalo de transmisión de los mensajes a 60 sg. En la gráfica inferior se observa que, aunque existen muchas fluctuaciones, en media ambos escenarios son similares, siendo el tiempo medio de recepción de los mensajes superior al del intervalo de 30sg.

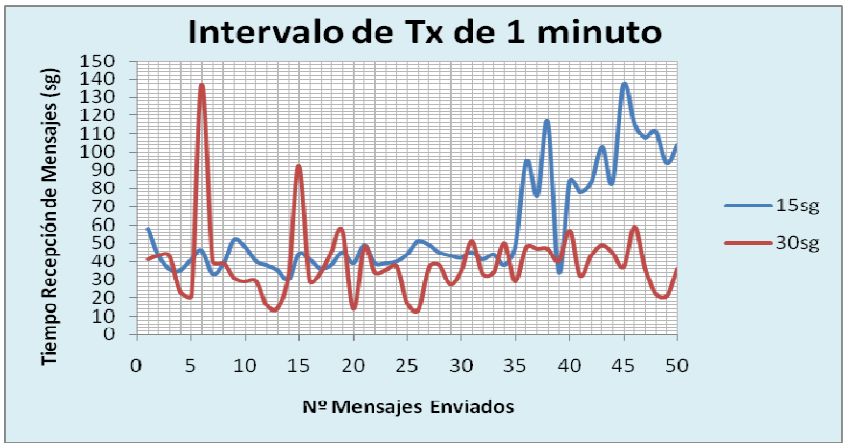


Figura 54: Comparativa Escenarios 1-4 de la simulación (Intervalo de Tx de 60sg)

- A continuación, lo que se ha fijado ha sido el *Timer* de chequeo del *Servidor* a 15sg y se ha ido variando el intervalo de transmisión de los mensajes por parte del *Cliente*:

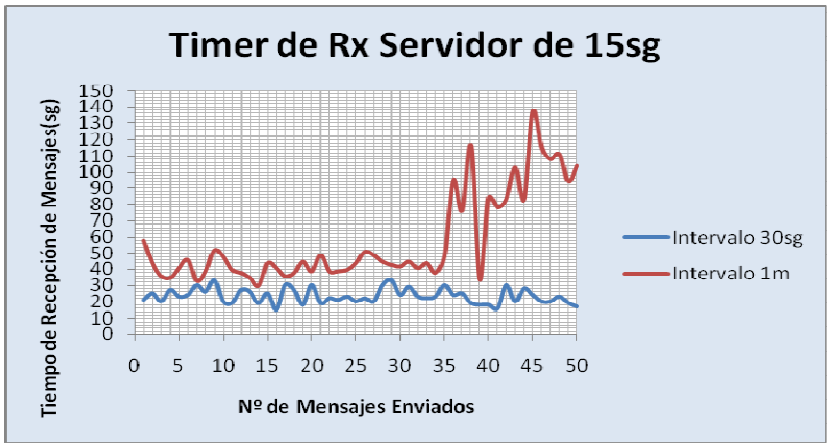


Figura 55: Comparativa Escenarios 1-4 de la simulación (*Timer* de chequeo del *Servidor* de 15sg)

- Se observa que el sistema funciona con mayor velocidad cuando, tanto la tasa de generación de mensajes como la de chequeo del *Servidor* se encuentran a su mínimo valor, ya que los mensajes enviados se procesan rápidamente y no se producen retardos en el envío de las respuestas a los clientes.

- En el siguiente apartado, se ha fijado el tiempo de transmisión de los mensajes por parte del cliente a 60sg y se verifica la influencia que tiene el valor del *Timer* de chequeo del *Servidor* en los tiempos de respuesta. Se puede ver, que en este caso no es muy relevante el tiempo que se le asigne al *Timer* del *Servidor*, ya que los tiempos medios son muy parecidos; el sistema actúa con la misma velocidad para ambos casos.

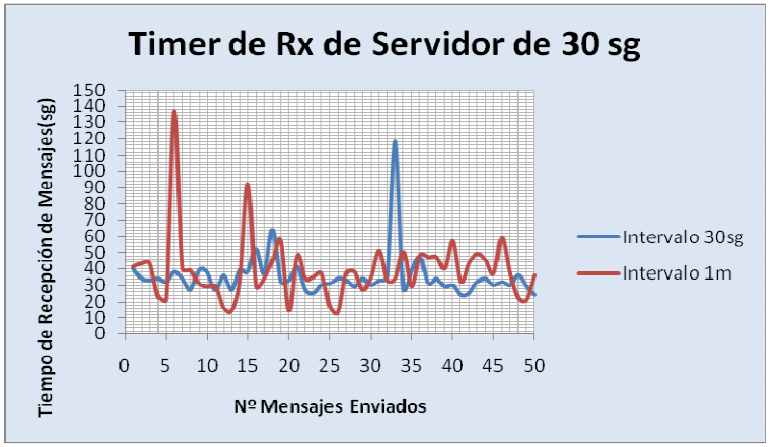


Figura 56: Comparativa Escenarios 1- 4 de la simulación (*Timer* de chequeo del *Servidor* de 60sg)

- Por último, se muestra una gráfica con los 4 escenarios posibles, donde se puede ver que el mejor caso se da cuando el intervalo de transmisión de los mensajes por parte del *Cliente* es de 30sg y el *Timer* de chequeo del *Servidor* es de 15sg. Mientras que el peor de los casos se da cuando el intervalo de transmisión de los mensajes es de 60sg y el *Timer* de recepción del *Servidor* es de 15sg.

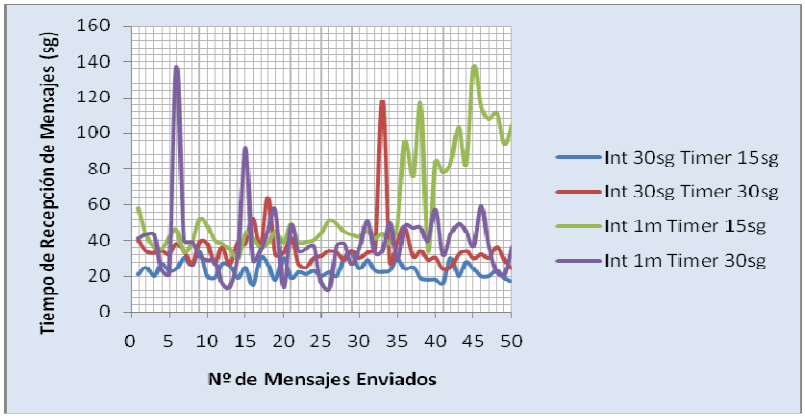


Figura 57: Comparativa Escenarios 1 – 4.

6.1.2 Simulaciones con 2 aplicaciónes Cliente

6.1.2.1 Escenario 5

- Condiciones:
  - Tasa de envío de la aplicación *Cliente* aleatoria en un intervalo de 30sg.
  - *Timer* de chequeo de la aplicación *Servidor* de 30sg.
  - *Timer* de chequeo de la aplicación *Cliente* de 15sg.

- Resultados de la simulación:

  - En la gráfica de abajo se puede comprobar, que en este caso el sistema es totalmente ineficiente, ya que los tiempos de recepción de los mensajes crecen exponencialmente a media que aumenta el número de envíos por parte de los *Clientes*; el buzón de entrada del *Servidor* se encuentra saturado, con lo que no puede dar salida inmediata a los mensajes de respuesta y los tiempos son muy elevados. Para solucionar esta deficiencia se ha implementado el escenario 6.

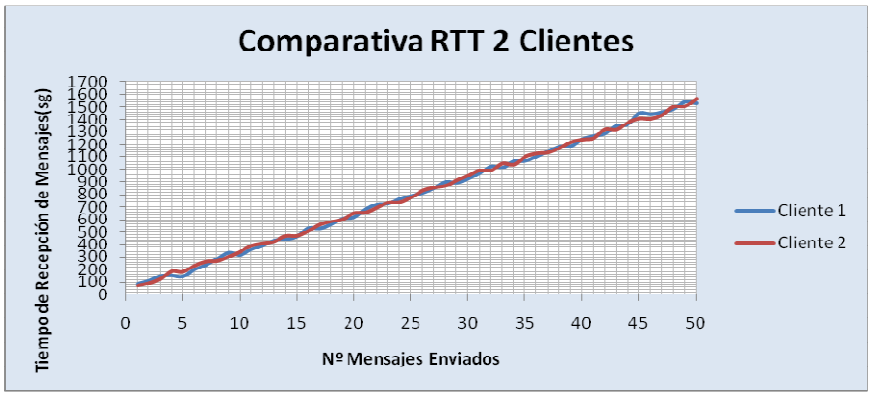


Figura 58: RTT del Escenario 5 de la simulación.

6.1.2.2 Escenario 6

- Condiciones:
  - Tasa de envío de la aplicación *Cliente* aleatoria en un intervalo de 30sg.
  - *Timer* de chequeo de la aplicación *Servidor* de 10sg.
  - *Timer* de chequeo de la aplicación *Cliente* de 10sg.

- Resultados de la simulación:

  - Se muestra la comparativa de los tiempos de recepción de los mensajes por parte de ambos *Clientes*, donde se observa que son similares; para el *Cliente* 1 es de 24.17sg, y para el *Cliente* 2 de 29.66sg. Esto es así debido a que todos los *Timers* de chequeo de todas las aplicaciones se han reducido al máximo, permitiendo una mayor velocidad de procesamiento y, por tanto, una reducción considerable de los tiempos de la comunicación.

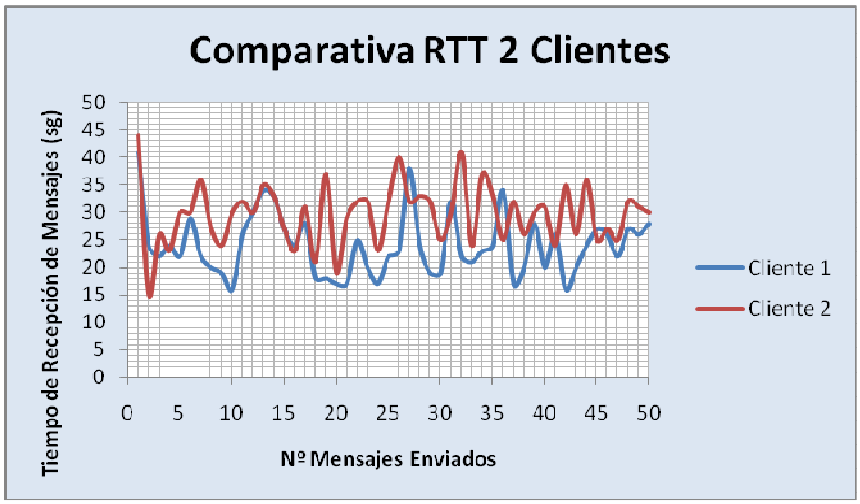


Figura 59: RTT del Escenario 6 de la simulación.

6.1.2.3 Escenario 7

- Condiciones:
  - Tasa de envío de la aplicación *Cliente* aleatoria en un intervalo de 60sg.
  - *Timer* de chequeo de la aplicación *Servidor* de 10sg.
  - *Timer* de chequeo de la aplicación *Cliente* de 10sg.

- Resultados de la simulación:
  - En este caso, al incrementar el intervalo de transmisión de mensajes por parte de los *Clientes* y a la reducción de los *Timers* a su mínimo valor, el sistema se encuentra más descargado de mensajes y, por tanto, se observa una notable mejora en los tiempos medios, para el *Cliente 1* es de 25.9sg y para el *Cliente 2* es de 26.3sg.

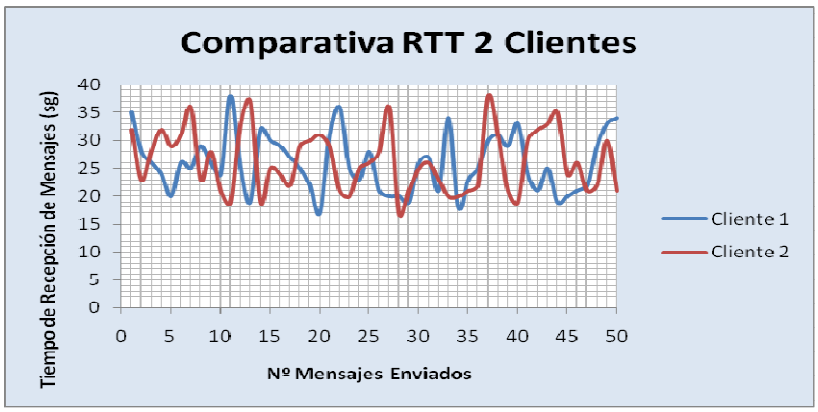


Figura 60: RTT del Escenario 7 de la simulación.

6.1.2.4 Escenario 8

- Condiciones:
  - Tasa de envío de la aplicación *Cliente* aleatoria en un intervalo de 60sg.
  - *Timer* de chequeo de la aplicación *Servidor* de 30sg.
  - *Timer* de chequeo de la aplicación *Cliente* de 15sg.

- Resultados de la simulación:
  - En este último caso, a pesar de volver a aumentar el valor de los *Timers* de chequeo, se observa que, aunque el tiempo crece, lo hace muy por debajo del caso expuesto en el escenario 5, ya que al ser la tasa de envío mayor, el *Servidor* es capaz de procesar los mensajes y enviar las respuestas a los *Clientes* en un tiempo menos ya que su buzón no se encuentra desbordado.

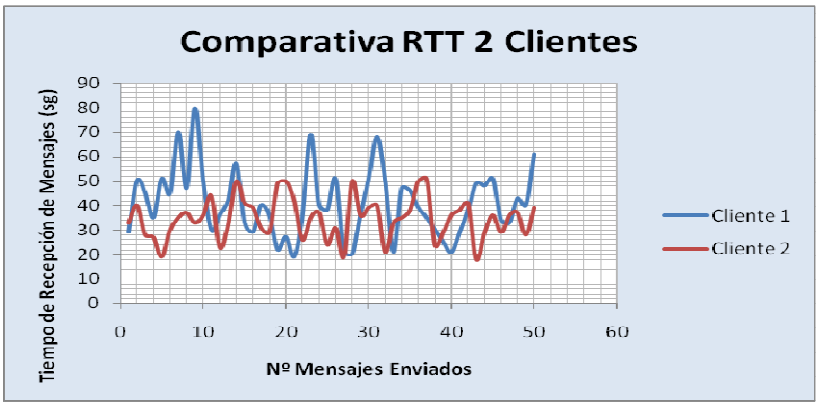


Figura 61: RTT del Escenario 8 de la simulación.

6.1.2.5 Comparativa de los escenarios 4 -8

- En la gráfica inferior se observa claramente que el peor de los escenarios se da cuando el intervalo de transmisión de los mensajes es de 30sg y el *Timer* de chequeo del *Servidor* se encuentra a 30sg. El mejor caso se da para un intervalo de transmisión de 60sg y un valor de los *Timers* de chequeo de 10sg.

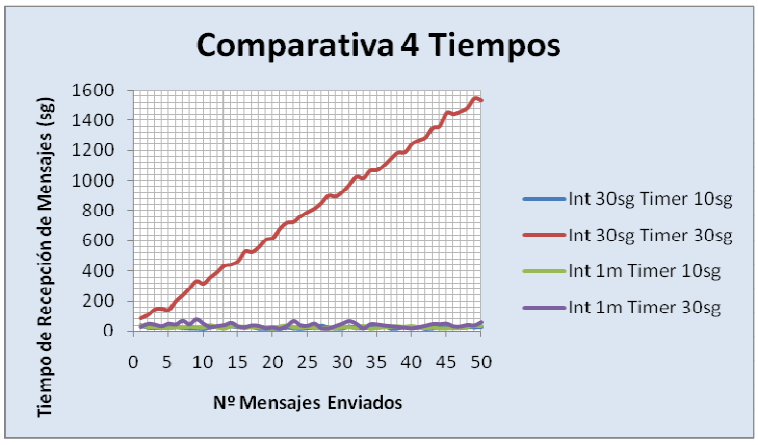


Figura 62: Comparativa Escenarios 4-8.

6.1.3 Conclusiones Extraídas de la simulación.

- En primer lugar, reseñar que el tamaño de los mensajes no ha influenciado en el retardo producido en el sistema, ya que al ser correos electrónicos formados sólo por cadenas de texto, su tamaño resulta irrelevante.

- En segundo lugar, se va comentar la influencia de los parámetros del sistema que influyen en los tiempos de transmisión de los mensajes:

    - **Intervalo de transmisión de los mensajes:** este tiempo, presente en las aplicaciones *Cliente* y que nos informa del periodo de tiempo en el que éstas emiten sus correos, influye de manera significativa en la velocidad de procesamiento, ya que cuanto menor sea su valor más correos llegarán al buzón de la aplicación *Servidor*, dando ésta una salida más lenta a los correos de respuesta. Por el contrario, si este tiempo es mayor, los correos llegan al *Servidor* de manera más escalonada y son procesados con mayor rapidez.

    - ***Timers* de chequeo de correo recibido en las aplicaciones *Cliente* y *Servidor*:** este es el principal parámetro a considerar a la hora del retardo del sistema, ya que determina la velocidad a la que se comprueba la llegada de correos a las aplicaciones. Por tanto, si se quiere que el sistema trabaje a una mayor velocidad y lo haga de forma eficiente, este tiempo se deberá reducir al mínimo.

    - **Número de Clientes en el sistema:** conforme se va aumentado el número de aplicaciones *Cliente* en el sistema, el número de mensajes en circulación también se incrementa y, por tanto, el buzón de la aplicación *Servidor* se encontrará más lleno y los mensajes de salida se producirán con un mayor retardo.

- Por lo tanto, estos tres parámetros influyen de manera conjunta en el correcto funcionamiento del sistema. Si se quiere una mayor velocidad y eficiencia habrá que reducir al máximo los valores de los *Timers* de chequeo de recepción de correo de todas las aplicaciones, a un tiempo mínimo de 10sg. Además el intervalo de transmisión de los mensajes por parte de los *Clientes* deberá ser lo suficientemente elevado para evitar la circulación continua de mensajes y el colapso del buzón del *Servidor*, esto es, alrededor de 60sg. Fijados los dos parámetros anteriores, el número de clientes no es influyente en el retardo del sistema, a menos que aumente en exceso su tamaño, lo ideal serían entre 5 y 10 clientes.

- Para finalizar se han añadido unas capturas de las aplicaciones implementadas, obtenidas en la parte final de la simulación.

- Aplicación *Cliente 1:*

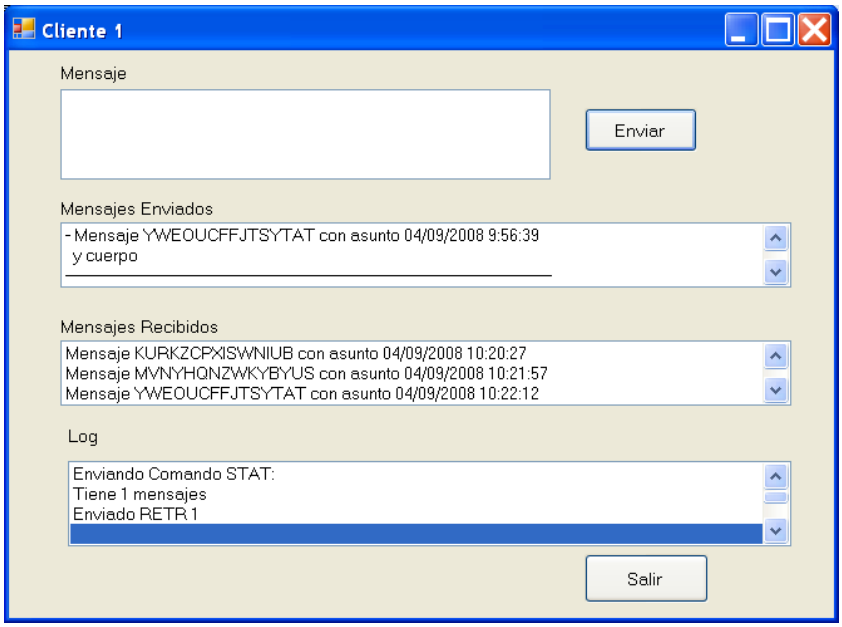


Figura 63: Estado de la aplicación *Cliente 1*durante la simulación.

- Aplicación *Cliente 2*:

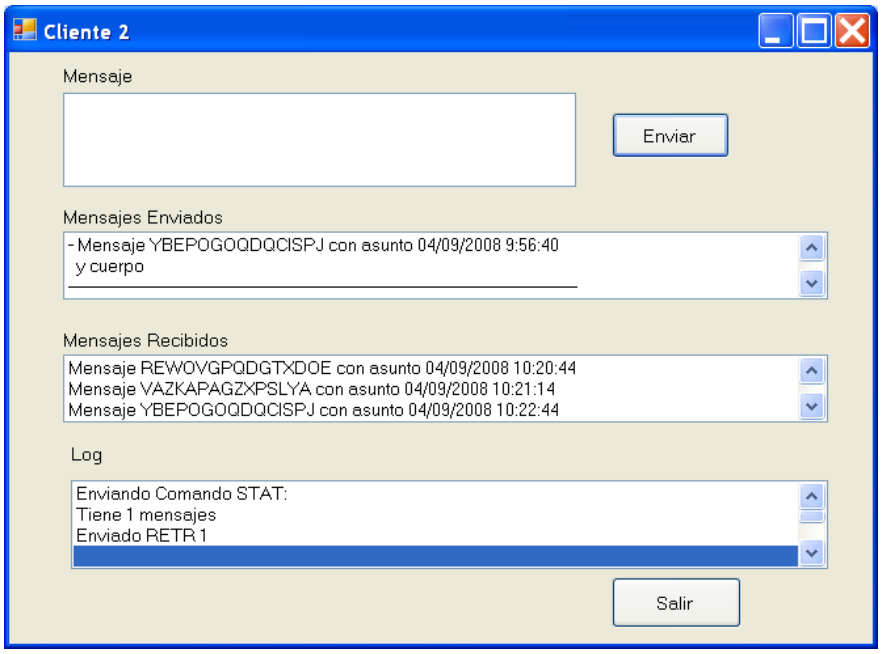


Figura 64: Estado de la aplicación *Cliente 2* durante la simulación.

- Aplicación *Servidor*:

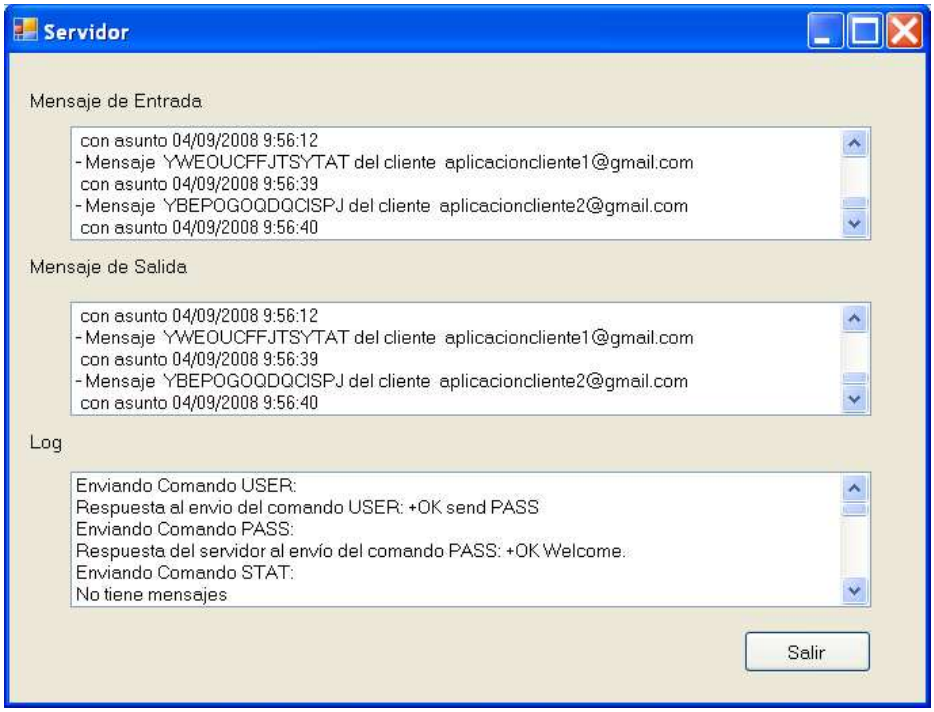


Figura 65: Estado de la aplicación *Servidor* durante la simulación.

Y el estado final de las cuentas de correo de todas las aplicaciones:

- Aplicación *Servidor*

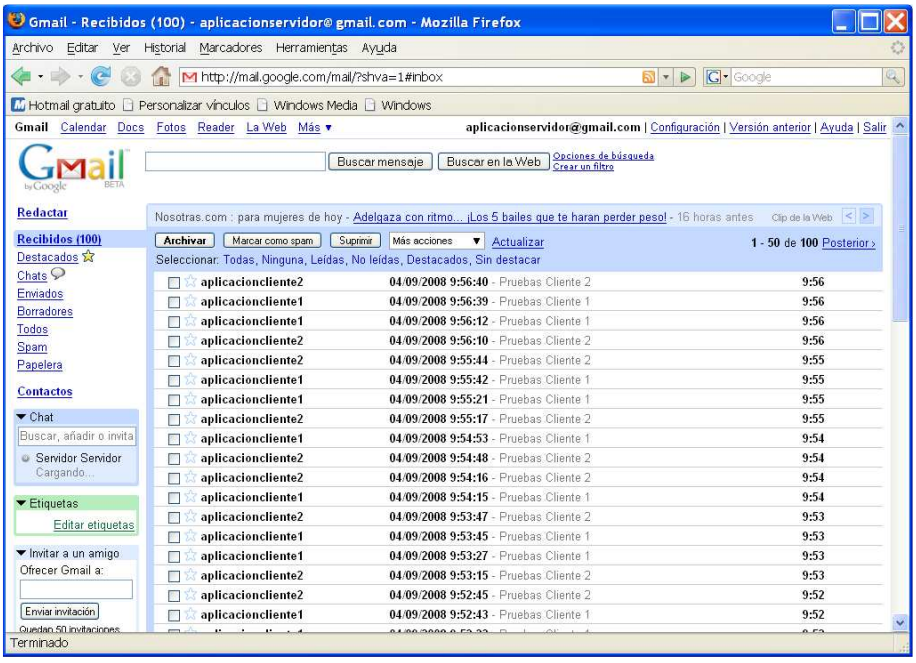


Figura 66: Estado al final de la simulación de la cuenta de correo de la aplicación *Servidor*.

- Aplicación *Cliente 1.*

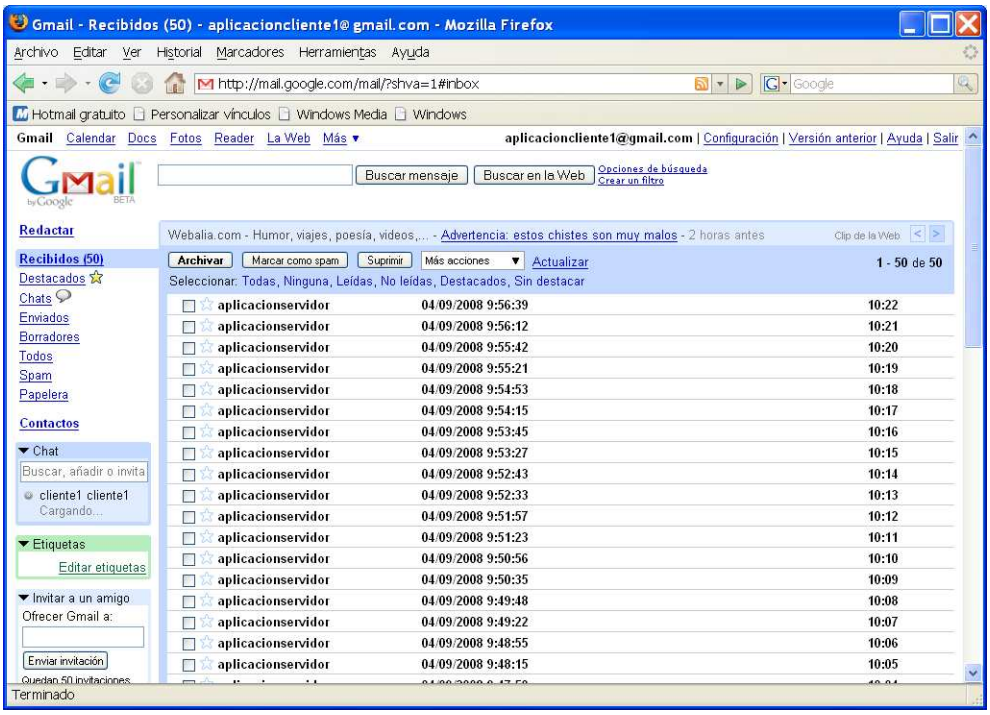


Figura 67: Estado al final de la simulación de la cuenta de correo de la aplicación *Cliente 1.*

- Aplicación *Cliente 2.*

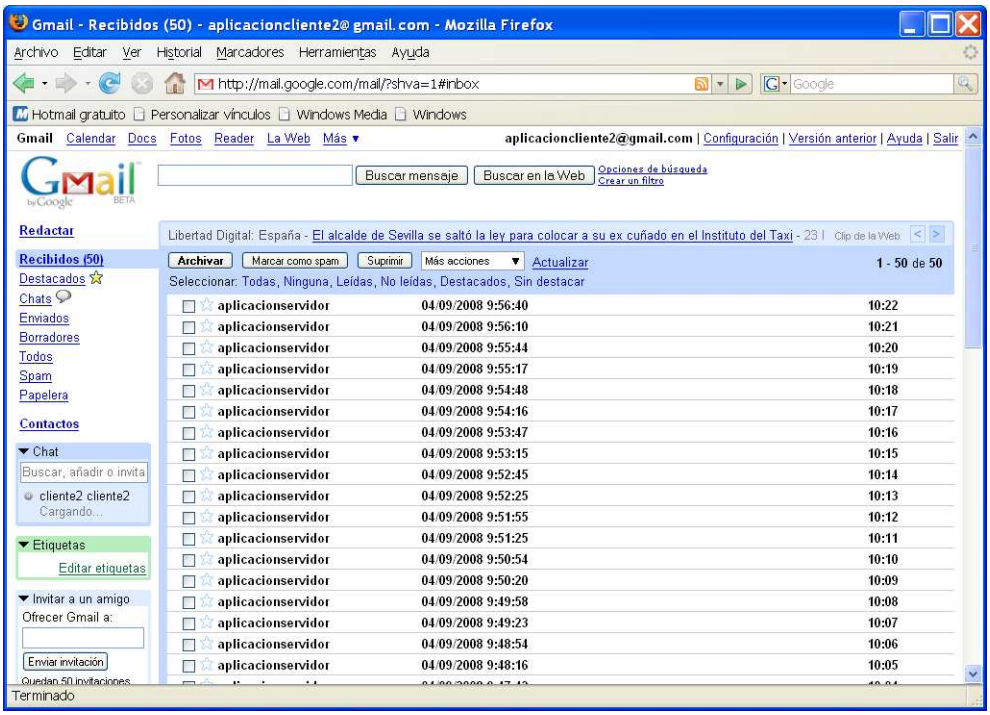


Figura 68: Estado al final de la simulación de la cuenta de correo de la aplicación *Cliente2.*

6.2 Conclusiones.

Una vez finalizado el proyecto, se va a presentar de forma resumida los pasos llevados a cabo para su realización, además de verificar si los objetivos planteados al comienzo del mismo han podido realizarse en su totalidad.

Para llevar a cabo este proyecto, en primer lugar se estudió de forma básica todo lo relacionado con el correo electrónico, lo que ha dado una visión general de algo que la mayoría de gente conoce.

Al estar este trabajo basado en clientes de correo, había que seleccionar un proveedor que cumpliera todos los requerimientos buscados (clientes de correo o correo *web*, POP o IMAP, seguridad,…), no se podía elegir una aleatoriamente, por lo que se realizó un exhaustivo estudio de los proveedores de correo gratuitos existentes en el mercado, optándose finalmente por *Gmail*.

A continuación llegó uno de los aspectos más complicados del proyecto, estudiar en profundidad los protocolos que intervienen en una comunicación de correo electrónico; para ello, se hizo uso de los RFCs, documentos en los que se explica detalladamente cada uno de los aspectos de cada protocolo:

- SMTP fue el primero en ser estudiado, su funcionamiento era sencillo, sin embargo, estaba formado por multitud de comandos con sus correspondientes códigos de respuesta, por lo que resultaba muy tedioso abarcar todas las posibilidades que ofrecía. Afortunadamente, el API de C, ofrece una librería llamada *System.Net.Mail*, que facilitó enormemente la tarea de implementar dicho protocolo, ya que mediante sencillas clases, métodos y propiedades se evitaba tener que utilizar todos los comandos existentes para enviar un correo electrónico.

- Para el caso del protocolo de descarga, no hubo la misma suerte, ya que los dos protocolos existentes, POP e IMAP, no estaban implementados en ninguna librería. Finalmente, se optó por emplear POP3, ya que era el que mejores posibilidades ofrecía. Si implementación fue tediosa, ya que enviar una serie de comandos, recibir sus respuestas y actuar en consecuencia, pero finalmente se logró el éxito.

El tema de la seguridad, ha sido estudiado en profundidad, ya que hoy en día es uno de los aspectos más importantes cuando se lleva a cabo una comunicación, y era de recibo dedicarle un amplio apartado.

Finalmente, se llevó a la práctica todo lo estudiado con anterioridad, resultando difícil el camino al principio, lo que resultó positivo para el proyectista, ya que dio la oportunidad de ampliar horizontes y agregar nuevas funcionalidades al proyecto que, posiblemente, si todo hubiera ido bien, no se hubieran llevado a la práctica.

Por lo tanto, como conclusión final, el autor de este proyecto ha cumplido con creces los objetivos planteados al comienzo de esta aventura y está satisfecho por el trabajo realizado.

6.3 Líneas de trabajo futuras.

Debido a que este proyecto está basado principalmente en la capa de comunicaciones que interconecta a las aplicaciones *Cliente* y *Servidor* con el servidor de correo de *Gmail*, una línea de trabajo sería adaptar la capa de aplicación de cada uno de los proyectos a actividad concreta, como puede ser la recogida de datos de un sensor para su procesado, o el envío automático de correos de notificación a los diferentes clientes conectados.

En este proyecto se ha empleado como protocolo de descarga de correos, POP3, debido a que su uso está extendido entre la mayoría de los proveedores de correo. Sin embargo, el estándar IMAP se está propagando de manera considerable, por lo que otra actuación futura sería implementar conjuntamente con POP3, el protocolo IMAP, abriendo así mayores posibilidades de acceder a todos los proveedores de correo.

Para evitar que el sistema se colapse, se podría implementar un control de flujo que llevara a cabo un control de los clientes presentes en el sistema y supervisara los intervalos de transmisión de los mensajes. Por último, habría que mejorar la batería de pruebas para ajustar mejor los parámetros y conocer los regímenes de funcionamientos estables del sistema.

Capítulo 7: BIBLIOGRAFÍA

7. Bibliografía

[1] Visual C# .NET:
Jason Price & Mike Gunderloy,
Ed. Sybex

[2] Pro C# 2005 and the .NET 2.0 Platform:
Andrew Troelsen,
Ed. Apress 3ª Edición

[3] Advanced .NET Remoting:
Ingo Rammer & Mario Szpustza,
Ed. Apress 2ª Edición

[4] Descripción de la herramienta de programación Microsoft Visual C# 2005 Express Edition
http://www.microsoft.com/spanish/msdn/vstudio/Express/VCS/default.mspx

[5] Ayuda y soporte técnico de *Gmail*: http://mail.google.com/support/
Configuración de clientes de correo POP3:
http://mail.google.com/support/bin/answer.py?answer=13287

[6] Comparativa de correos *web* gratuitos:
http://www.consumer.es/*web*/es/tecnologia/internet/2005/05/25/142236.php

[7] RFC 2821 Simple Mail Transfer Protocol

[8] RFC 1939 Post Office protocol- Version 3

[9] RFC 3501 Internet Message Access Protocol Version 4rev1.

[10] RFC 1521 - MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies.

[11] RFC 2046 Multipurpose Internet Mail Extensions (MIME) Part Two Media Types.

[12] RFC 2047 MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text.

ANEXOS

Anexo I. Clase SslStream

- Constructor público:
  - *SslStream*: Inicializa una nueva instancia de la clase *SslStream.*

- Métodos públicos:
  - *AuthenticateAsClient:* Autentica el cliente de una conexión cliente-servidor.
  - *AuthenticateAsServer:* Lo llaman los servidores para autenticar el servidor, y opcionalmente el cliente, en una conexión cliente-servidor.
  - *Flush:* Provoca la escritura de los datos almacenados en en el búfer del dispositivo subyacente.
  - *GetHashCode:* Sirve como función hash para un tipo concreto. *GetHashCode* es apropiado para su utilización en algoritmos de hash y en estructuras de datos como las tablas hash.
  - *Read*: Lee datos de esta secuencia y los almacena en la matriz especificada.

- Propiedades públicas:
  - *CanRead:* Obtiene un valor *Boolean* que indica si se puede leer la secuencia subyacente.
  - *CanTimeout:* Obtiene un valor *Boolean* que indica si la secuencia subyacente admite tiempos de espera.
  - *CheckCertRevocationStatus:* Obtiene o establece un valor *Boolean* que indica si la lista de revocación de certificados se coteja durante el proceso de validación de certificados.
  - *CipherAlgorithm*: Obtiene un valor que identifica el algoritmo de cifrado masivo utilizado por esta secuencia *SslStream*.
  - *HashAlgorithm:* Obtiene el algoritmo utilizado para generar códigos de autenticación de mensajes (MAC).
  - *IsAuthenticated:* Obtiene un valor *Boolean* que indica si la autenticación se realizó correctamente.
  - *IsEncrypted*: Obtiene un valor de *Boolean* que indica si esta secuencia *SslStream* utiliza cifrado de datos.
  - *IsMutuallyAuthenticated:* Obtiene un valor de *Boolean* que indica si se autenticaron tanto el cliente como el servidor.
  - *IsServer:* Obtiene un valor de *Boolean* que indica si la parte local de la conexión utilizada por esta secuencia *SslStream* se autenticó como servidor.
  - *KeyExchangeAlgorithm:* Obtiene el algoritmo de intercambio de claves utilizado por esta *SslStream*.
  - *LocalCertificate:* Obtiene el certificado utilizado para autenticar el extremo local.
  - *RemoteCertificate:* Obtiene el certificado utilizado para autenticar el extremo remoto.
  - *SslProtocol:* Obtiene un valor que indica el protocolo de seguridad utilizado para autenticar esta conexión.

Anexo II. Clase SmtpClient.

Como propiedades más destacadas tenemos las siguientes:

- *ClientCertificates:* Especifique los certificados que deben utilizarse para establecer la conexión SSL (Secure Sockets Layer).
- *Credentials:* Obtiene o establece las credenciales utilizadas para autenticar al remitente.
- *DeliveryMethod:* Especifica la forma en que se controlarán los mensajes de correo electrónico salientes.
- *EnableSSL:* Especifique si el objeto SmtpClient utiliza SSL (Secure Sockets Layer) para cifrar la conexión.
- *Host:* Obtiene o establece el nombre o la dirección IP del host que se utiliza para las transacciones SMTP.
- *Port:* Obtiene o establece el puerto utilizado para las transacciones SMTP.
- *Timeout:* Obtiene o establece un valor que especifica el intervalo de tiempo a partir del cual se considera que una llamada a Send sincrónica excede el tiempo de espera.

Como métodos más destacados tenemos:

- *Send:* Envía el mensaje especificado a un servidor SMTP para su entrega.
- *SendAsync:* Envía el mensaje de correo electrónico especificado a un servidor SMTP para su entrega. Este método no bloquea el subproceso que realiza la llamada y permite al llamador pasar un objeto al método que se invoca cuando termina la operación.
- *SendAsyncCancel:* Cancela una operación asincrónica para enviar un mensaje de correo electrónico.
- *OnSendCompleted:* Provoca el evento SendCompleted.

Anexo III. Clase MailAddress.

Se utiliza la clase MailAddress para almacenar la información de dirección de los mensajes de correo electrónico.

Como constructor de esta clase, se utiliza la siguiente sentencia que obtiene de forma individual las direcciones del origen y el destinatario de los mensajes: *public MailAddress (string address).*

Las propiedades más destacadas de esta clase son las siguientes:

- *Address:* Obtiene la dirección de correo electrónico que se especificó al crear esta instancia.
- *Host:* Obtiene la parte correspondiente al host de la dirección que se especificó al crear esta instancia.
- *User:* Obtiene la información de usuario a partir de la dirección que se especificó al crear esta instancia.

Anexo IV. Clase MailMessage.

Se utilizan instancias de la clase *MailMessage* para crear mensajes de correo electrónico que se transmiten a un servidor SMTP para entregarse mediante la clase *SmtpClient*.

Se ha utilizado el siguiente constructor de clase: *public MailMessage (MailAddress origen, MailAddress destino),* el cuál contiene como argumentos las direcciones origen y destino obtenidas mediante la clase anterior *MailAddress*.

Una vez declarado y creado el objeto mediante la sentencia procedemos a asignar el valor a diversas propiedades de esta clase, como son:

- *CC* (Copias): Obtiene la colección de direcciones que contiene los destinatarios para recibir copia (CC) de este mensaje de correo electrónico.
- *CCO* (Copias ocultas): contiene los destinatarios ocultos de copia (CCO) de este mensaje de correo electrónico.
- *Attachments:* Obtiene la colección de datos adjuntos que se utiliza para almacenar los datos adjuntos a este mensaje de correo electrónico.
- *Subject:* Obtiene o establece la línea de asunto de este mensaje de correo electrónico.
- *Body*: Valor de tipo String que contiene el texto del cuerpo.
- *BodyEncoding:* Obtiene o establece la codificación que se utiliza para codificar el cuerpo del mensaje.
- *DeliveryNotificationOptions:* Obtiene o establece las notificaciones de entrega este mensaje de correo electrónico.
- *Headers:* Obtiene los encabezados de correo electrónico que se transmiten con este mensaje de correo electrónico.
- *IsBodyHTML:* Obtiene o establece un valor que indica si el cuerpo del mensaje de correo está en HTML.
- *Priority:* Obtiene o establece la prioridad del mensaje de correo electrónico.
- *ReplyTo:* Obtiene o establece la dirección de origen o de respuesta del mensaje de correo.
- *Sender:* Obtiene o establece la dirección del remitente de este mensaje de correo electrónico.