Universidad Nacional Autónoma de México

Facultad de Ingeniería

"Obtención de *Spam* en buzones de correo electrónico, cuyo protocolo de almacenamiento sea *POP3* o *IMAP*"

TESIS PROFESIONAL
QUE PARA OBTENER EL TÍTULO DE
INGENIERO EN COMPUTACIÓN

PRESENTA:
CARLOS SÁNCHEZ SILVA

DIRECTOR DE TESIS:
ING. JUAN JOSÉ CARREÓN GRANADOS

CIUDAD UNIVESITARIA, MÉXICO D.F. **Junio de 2004**

ÍNDICE

Introducción

El correo electrónico es un prominente medio de comunicación, el cual permite tanto el envío como la recepción de archivos. Para quienes utilizan o han utilizado el correo electrónico saben que es posible enviar documentos a cualquier persona en cuyo país esté disponible el servicio de Internet.

Básicamente para utilizar el correo electrónico sólo se requiere tener una computadora conectada a la red Internet y obtener una cuenta de correo electrónico, la cual es posible obtener mediante un pago por el servicio a algún *ISP*, o incluso de manera gratuita(con algunos proveedores de servicios de Internet como *hotmail, yahoo, terrra,* etcétera).
Obtener una cuenta de correo electrónico le permite al dueño de la misma utilizar tanto el servidor de correo saliente, como el servidor de correo entrante del *ISP* que le proporcionó la cuenta. El servidor de correo saliente es utilizado por el usuario para enviar mensajes de correo electrónico. Utilizar el servidor de correo entrante implica, la asignación de cierta porción del espacio disponible en el medio de almacenamiento utilizado por dicho servidor al usuario, para que él pueda recibir los mensajes de correo electrónico enviados a su cuenta.

Desafortunadamente no todos los *ISP* proporcionan cuentas de correo electrónico con capacidad ilimitada de almacenamiento, por lo cual si el espacio asignado para almacenar mensajes de un usuario ha sido completado, no podrá recibir más mensajes, hasta que borre algunos.

Actualmente algunas empresas o personas envían mensajes de correo electrónico, con fines de lucro, información, e incluso de fraude, a miles de usuarios de correo electrónico, quienes nos les han solicitado información, a esta situación se le ha llamado *Spam*[1]. Una característica que distingue cuales mensajes pueden considerarse correo electrónico no solicitado, se puede obtener con los datos tanto de encabezado del mensaje como con los de remitente.

Es posible enviar mensajes con datos falsos de remitente, sin embargo los datos de encabezado indican cuál servidor de correo saliente ha sido utilizado para enviar el mensaje, en caso de no coincidir los dominios de ambos (dominio del servidor de correo saliente y dominio de la cuenta de correo electrónico del remitente) es posible inferir que los datos de remitente son falsos.

En el presente trabajo se desarrolló una aplicación que permite al usuario conocer cuáles de sus mensajes de correo electrónico podrían tener falsos datos de remitente, además proporciona un conjunto de opciones para filtrar su mensajes.

Este trabajo está estructurado en 10 capítulos, el capítulo 1 corresponden a la definición del problema. A continuación se describe el contenido de los siguientes capítulos:

[1] Spam: anglicismo cuya connotación es repugnancia.

En el capítulo 2: Correo Electrónico, se describe de manera breve el origen del correo electrónico indicando que es un servicio de Internet, por lo cual también se incluye una descripción sobre su historia.

Dentro del capítulo 3: Protocolos de correo electrónico, se especifican los protocolos tanto de almacenamiento(*POP3* e *IMAP*) como de transporte(*SMTP*), describiendo que servicios ofrecen y por que son necesarios. También se explica el procedimiento utilizado por cada uno de estos para efectuar su función, indicando comandos, respuestas y algunos mensajes de error.

En el capitulo 4: *API JavaMail* y *JavaBeans,* son descritas las herramientas utilizadas para el desarrollo del sistema. El objetivo de este capítulo es exponer un fragmento de las clases esenciales de la *API*, además de mostrar mediante un esquema la manera de interactuar de la *API* y los *JavaBeans* con los protocolos de correo electrónico.

El capítulo 5: *Spam;* trata el tema de correo electrónico no solicitado(*Spam*), también son descritas algunas consecuencias del mismo, para lo cual se utilizaron los resultados de un estudio llevado a cabo por American Pew[2]. Adicionalmente se exponen algunos recursos tanto tecnológicos como legales para combatir el *Spam*.

En el capítulo 6: Análisis y Diseño, se trata el tema de la metodología utilizada para el desarrollo del sistema, por lo cual son descritos los requerimientos del sistema incluyendo el diagrama de casos de uso. En la parte de análisis son diseñados los diagramas de secuencia, clases y estados. Finalmente en la parte de diseño se expone el diagrama de componentes.

En el capítulo 7: Programación del sistema, únicamente se muestra un fragmento con lo más importante del sistema, el resto del código fuente se omite por que es demasiado extenso.

En el capítulo 8: Instalación y pruebas, se describen cuáles son los requerimientos de *software* necesarios para utilizar el sistema, y donde se pueden obtener.

Finalmente en el capítulo 9 de manera breve se exponen las conclusiones; y el capítulo 10 está dedicado a la parte de bibliografía.

[2] Organización creada por el Research Center, cuyo objetivo es explorar el impacto de Internet en la sociedad.

1. Definición del problema

El protocolo *SMTP* define los procedimientos necesarios para el envío de correo electrónico a través de redes(*Internet* y redes *Intranet* que utilicen los protocolos *TCP* e *IP*), los cuales están descritos en la *RFC 8*21, la cual fue publicada en el año de 1982.

Resulta sencillo implementar interfaces que lleven a cabo los procedimientos para envío de correo electrónico; con relativa facilidad es posible enviar masivamente correo electrónico a las cuentas de usuarios. De esta manera cada usuario de correo electrónico está expuesto a recibir mensajes con remitente desconocido cuyo contenido es engañoso o deshonesto (puede ser con fines de lucro, fraude e incluso ofensivo), lo cual se ha denominado con el nombre de *Spam*.

Las especificaciones descritas en la *RFC 821* no están diseñadas para identificar la identidad del remitente, es posible recibir mensajes de correo electrónico con remitente falso(a nombre de otra persona u organización).

En el mes de febrero del año 1999, el grupo de trabajo en red publica la *RFC 2505* "Recomendaciones *Anti-Spam* para servidores *SMTP* de transferencia de correo", con el objetivo de manejar y prevenir el correo electrónico no deseado. Algunos días después, en el mes de marzo es publicada la *RFC 2554* "Extensión del Servicio de *SMTP* para Autenticación", en el cual se define un servicio para que el servidor de correo saliente (servidor *SMTP*) acepte o rechace peticiones para envío de correo electrónico; por lo cual únicamente usuarios registrados podrían utilizar el servidor.

En el mes de abril del año 2001 es publicada la *RFC 2821*, "Protocolo Simple de Transferencia de Correo", documento que actualiza la *RFC 821*. En esta especificación se describe en la sección consideraciones de seguridad que el correo enviado a través del protocolo *SMTP* es inherentemente inseguro, ya que los usuarios pueden negociar directamente con servidores *SMTP* para el envío de mensajes a través de Internet.

En la sección 2.2 de la *RFC 2821* ("El modelo de extensión") se indica que si el soporte para los servicios de autenticación es añadido, se debe llevar a cabo de tal manera que permita a implementaciones anteriores continuar trabajando aceptablemente, también se recomienda analizar costos de interoperatividad antes de implementar extensiones en los servidores de correo saliente, incluso se afirma: "... e*n muchos casos el costo de extender el servicio SMTP probablemente sobreestima el beneficio*[1]*...*".

Ante esta situación la responsabilidad de manejar el correo electrónico no solicitado, es asignada implícitamente a los agentes de correo de usuario(interfaces para administrar el correo electrónico), de esta manera cada usuario puede diseñar filtros para administrar sus propios mensajes, sin embargo sólo algunas interfaces de correo electrónico permiten automatizar tareas de eliminar o enviar mensajes.

[1] *RFC 2821* página 6.

Las empresas que ofrecen cuentas de correo electrónico proporcionan alternativas para evitar que el correo electrónico no solicitado llegue a sus usuarios, las más útiles son;

1. Bloqueo de direcciones de correo electrónico: el usuario especifica direcciones de las cuales no desea recibir mensajes. También es posible bloquear dominios; por ejemplo especificar que sea eliminado cualquier correo cuyo dominio sea *prodigy.net.mx*.

2. Eliminación de correo electrónico cuando el remitente sea considerado como desconocido para el dueño de la cuenta de correo(éste ultimo indicará de que personas no eliminará el correo sin antes leerlo).

3. Filtros para manejar el correo(eliminar, contestar, enviarlo a alguna carpeta ...) de acuerdo con los datos de encabezado indicados en cada correo.

Una investigación llevada a cabo por *Pew Internet & American Life Project*(organizaciones creadas por iniciativa de el *Pew Research Center*, cuyo objetivo es explorar el impacto de el Internet sobre la sociedad) del 10 al 24 de Junio de 2003 para conocer los efectos de *Spam* sobre el correo electrónico indica (de entre muchos otros aspectos) lo siguiente:

- El 52% de usuarios de correo electrónico dice que el *Spam* los ha hecho confiar menos en este servicio de Internet.

- El 80% de usuarios de correo electrónico es molestado por el contenido deshonesto de *Spam*.

- La carga de *Spam* es mayor sobre cuentas de correo electrónico de sistemas basados en estándares abiertos.

- El 76% de usuarios de correo electrónico es molestado por el contenido ofensivo u obsceno de *Spam*.

2. Internet y Correo Electrónico

ARPA(Agencia de Proyectos de Investigación Avanzada), comenzó a trabajar con una tecnología de red de redes a mediados de los años setenta; su arquitectura y protocolos tomaron su forma final entre 1977 y 1979.

La tecnología desarrollada por ARPA, incluye un grupo de estándares de red que especifican los detalles de cómo se comunican las computadoras, así como un grupo de reglas para interconectar redes y para *rutear* el tráfico. Conocido de manera oficial como el grupo de protocolos *TCP/IP*, éste puede utilizarse para comunicarse a través de cualquier grupo de redes interconectadas. Por ejemplo, algunas empresas utilizan el *TCP/IP* para interconectar todas las redes dentro de la corporación, aun cuando las empresas no tengan una conexión hacia redes externas.

Protocolos como el *TCP* y el *IP* proporcionan las reglas para la comunicación. Contienen los detalles referentes a los formatos de los mensajes, describen cómo responde una computadora cuando llega un mensaje y especifican de qué manera una computadora maneja un error u otras condiciones anormales. En cierto sentido, los protocolos son para las comunicaciones los que los algoritmos son para la computación. Un algoritmo permite especificar o entender un cómputo aunque no se conozcan los detalles de un juego de instrucciones de *CPU*. De manera similar, un protocolo de comunicaciones permite especificar o entender la comunicación de datos sin depender de un conocimiento detallado de una marca en particular de hardware de red.

La tecnología que dio origen a Internet permitió diseñar nuevos protocolos que proporcionan servicios bajo el paradigma *cliente-servidor*, un ejemplo es el correo electrónico.

La idea del correo electrónico fue proporcionar un servicio que permitiera a cada usuario enviar mensajes, y documentos adjuntos utilizando algún servidor de correo saliente(para ello se requirió diseñar el protocolo *SMTP*) hacia otros usuarios de correo electrónico, y utilizar algún servidor de correo entrante para obtener los mensajes que le han sido enviados (para ello fue necesario diseñar el protocolo *POP*; posteriormente se diseñó el protocolo *IMAP*). Cada servicio proporcionado a través de Internet utiliza protocolos específicos para ello.

2.1 Administración de Internet(IAB, IRTF, IETF)

Internet Architecture Board (Junta de Arquitectura de Internet). Debido a que el grupo de protocolos *TCP/IP* para red de redes no surgió de una manera específica o a través de alguna sociedad profesional reconocida, es conformada en 1983 la (IAB) con el objetivo de proporcionar el enfoque y coordinación para gran parte de la investigación y desarrollo subyacentes de los protocolos *TCP/IP*, y también guiar la evolución de Internet. Seis años después la IAB pasó de ser un grupo de investigación específica de ARPA a una organización autónoma. Cada miembro de la IAB dirigía una de las diez FTI(Fuerza de Tarea de Internet), asignada a investigar un problema o un grupo de aspectos importantes.

La IAB se reunía varias veces al año para escuchar reportes de estado de cada fuerza de tarea, escuchar y revisar directivas técnicas, discutir políticas e intercambiar información con los representantes de ARPA y NSF, quienes proporcionaban los fondos para las operaciones y la investigación de Internet.

Para el verano de 1989, la tecnología *TCP/IP* así como Internet habían crecido más allá del proyecto inicial de investigación y se convirtieron en medios de producción de los que miles de personas dependen para sus negocios diarios; los investigadores que diseñaron y vieron cómo se desarrolló el *TCP/IP* se encontraron rebasados por el éxito comercial de su creación. Para reflejar la realidad política y comercial tanto del *TCP/IP* como de Internet, la IAB fue reorganizada, ver figura 2.1.

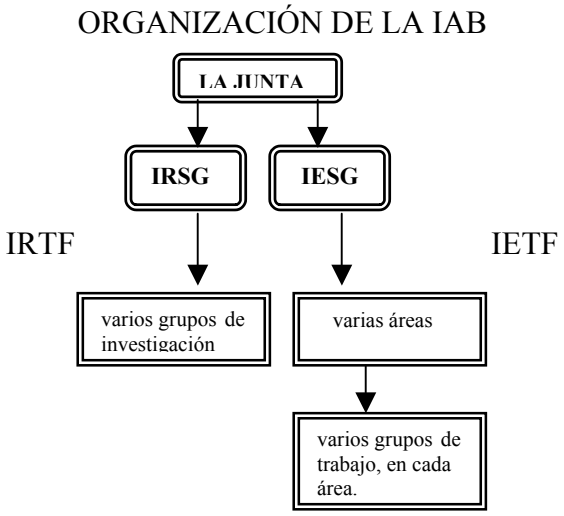


Figura 2.1Estructura de la IAB después de la reorganización de 1989

Después de la reorganización, la IAB contiene dos grupos mayores: la Fuerza de Tarea de Investigación Internet(IRTF) y la Fuerza de Tarea de Ingeniería Internet(IETF); el primero de estos coordina las actividades de investigación relacionadas con los protocolos *TCP/IP* y con la arquitectura de red de redes en general. El segundo grupo se concentra en problemas de ingeniería a corto y mediano plazo.

El presidente de la IETF y los gerentes de área forman el Grupo de Control de Ingeniería Internet(IESG), que son las personas responsables de coordinar los esfuerzos de los grupos de trabajo de la IETF.

El Grupo de Control de Investigación de Internet(IRSG) establece prioridades y coordina las actividades de investigación.

2.1.1 Solicitud de comentarios de Internet (RFC)

Ninguna marca es dueña de la tecnología *TCP/IP*, por lo cual la documentación de protocolos, estándares y políticas no se pueden obtener con un distribuidor. La Fundación Nacional de Ciencias proporciona fondos aun grupo en AT&T para que mantenga y distribuya información sobre el *TCP/IP*. Conocido como Centro de Información de la Red Internet (INTERNIC), maneja muchos detalles administrativos para Internet, además de distribuir la documentación.

La documentación de trabajo en Internet, las propuestas para protocolos nuevos o revisados, así como los estándares del protocolo *TCP/IP*, aparecen en una serie de reportes técnicos llamados Solicitudes de Comentarios de Internet o *RFC*. El editor del *RFC* es un miembro de la IAB. La serie *RFC* está numerada en forma secuencial en el orden cronológico en que se escriben. Cada *RFC*, nuevo o revisado, tiene un número nuevo.

2.2 Correo electrónico

En la figura 2.2, se muestra un esquema lógico que indica el procedimiento con el cual se accede a la red Internet, en la imagen se indica que tanto usuarios como organizaciones requieren contratar el servicio de conexión a Internet a través de un *ISP*, quien les ofrecerá la conexión requerida de acuerdo a sus necesidades; el tamaño y giro de la empresa determinarán cuantos empleados necesitan conectar sus computadoras a la red Internet; el usuario generalmente sólo requiere conectar una computadora.

Actualmente las empresas desarrollan redes locales para compartir sus recursos, el nombre de una red cuyo funcionamiento sea exclusivo para una empresa u organización es *Intranet*.

El correo electrónico basado en estándares abiertos es un servicio de Internet, por lo tanto se requiere conexión a Internet para utilizarlo.

2.2.1 ¿Cómo funciona el correo electrónico?

En el diagrama de la figura 2.2 se observa un recuadro con la etiqueta - *Correo Electrónico* - , dentro de este se encuentran dos elementos cuyas etiquetas indican: Servidor de correo saliente y Servidor de correo entrante; cuando el usuario quiere enviar un mensaje por correo electrónico requiere:

1. Utilizar una computadora con acceso a Internet.
2. Solicitar a algún servidor de correo saliente que le envíe su mensaje (previamente indicará el mensaje, los datos adjuntos y los destinatarios).

Cuando el usuario quiere leer sus mensajes de correo electrónico necesita:

3. Utilizar una computadora con acceso a Internet.
4. Solicitar al servidor de corre saliente que tenga sus mensajes le permita acceder a estos.

Para que el usuario pueda satisfacer el punto 1 y 3, necesita el servicio de conexión a Internet, dicho servicio lo deberá contratar con algún *ISP*.

El punto 2 requiere que el usuario se comunique con el servidor de correo saliente, el cual sólo entiende peticiones mediante el protocolo de transporte *SMTP*(Simple Mail Transfer Protocol) , definido en la *RFC* 821 y 2821.

Para que el usuario se comunique con el servidor de correo entrante(lugar donde están almacenados sus mensajes) necesita comunicarse mediante el protocolo *POP*(POST OFFICE PROTOCOL), o *IMAP*(Internet Messsage Acces Protocol), o con algún otro protocolo de almacenamiento definido para sistemas de correo electrónico basado en estándares abiertos.

Afortunadamente cuando contratamos el servicio de conexión a Internet con algún *ISP*, este provee una o más *cuentas de correo electrónico.* Cuando tenemos una *cuenta de correo electrónico* podemos utilizar tanto el servidor de correo saliente como el de correo entrante, además cuando el usuario necesita utilizarlos accede a una interfaz gráfica donde simplemente selecciona diversas opciones que le permiten enviar mensajes y archivos adjuntos por correo electrónico, así como leer los mensajes que le han sido enviados.

El protocolo de transporte *SMTP* implementado en el servidor de correo saliente, se encarga de enviar los mensajes que el usuario indique, básicamente busca a través de Internet el servidor de correo entrante del destinatario para que este lo reciba.

El protocolo *POP o IMAP* o algún otro que sea de almacenamiento implementado en el servidor de correo entrante, recibirá los mensajes que lleguen a dicho servidor mediante algún protocolo de transporte.

¿Cómo identifica el servidor de correo saliente al servidor de correo entrante del destinatario ?

Cuando el *ISP* le asigna a algún usuario una cuenta de correo electrónico esta incluye una dirección de correo electrónico, por lo tanto todo usuario del correo electrónico tiene asociada una dirección de correo electrónico.

Cuando el usuario envía una mensaje debe indicar la dirección de correo electrónico del destinatario, con este dato y con otros procedimientos descritos en la *RFC* 821 y en su actualización *RFC* 2821 el mensaje y los datos adjuntos llegan a su destino.

Tanto el servidor de correo entrante como el de correo saliente son computadoras con *hardware* especifico, que implementan lo definido en los estándares mencionados anteriormente(*SMTP, POP, IMAP* principalmente) para atender las peticiones de los usuarios.

2.2.2 Sistemas de correo propietarios

Con el advenimiento de la conectividad de redes basada en *PC* a principios y mediados de los años ochenta, varios fabricantes empezaron a construir sistemas de correo electrónico que operaban en diferentes ambientes de computadoras personales.

Los paquetes propietarios de correo electrónico para las redes de computadoras personales generalmente son fáciles de instalar y de configurar. Los sistemas de correo propietarios basados en redes de *PC* de diferentes fabricantes no operan entre sí sin la ayuda de una *puerta de enlace de correo*, una computadora que convierte el correo de un formato propietario a otro. Los sistemas propietarios de correo no pueden *rutear* correo directamente a Internet. Para llevar a cabo dicha tarea requieren de otra puerta de enlace de correo para convertir el correo al formato *SMTP* (Protocolo Simple de Transporte de Correo), la sección del grupo de protocolos *TCP/IP* que maneja el *ruteo* del correo.

La principal desventaja de utilizar compuertas de correo es que reducen la confiabilidad. Una vez que la máquina del emisor transfiere un mensaje a la primera máquina intermedia, se descarta la copia local. Así, mientras el mensaje está en transito, tanto el receptor como el emisor carecen de alguna copia del mensaje. Si se dan fallas entre las máquinas intermedias puede provocar una pérdida del mensaje sin que informe al emisor.

2.2.3 Sistemas de correo electrónico basado en estándares abiertos

La IETF(Fuerza de trabajo de la Ingeniería de Internet) ha desarrollado estándares no propietarios para el funcionamiento del correo electrónico. Los estándares de correo de Internet son muy sencillos; siempre y cuando alguna red utilice el protocolo *TCP/IP*, es posible utilizar el correo de Internet; todo lo que se necesita es un servidor para correr *software* de aplicación del correo del lado del servidor y el software que corra en el cliente para acceder al correo del servidor.

Hay gran cantidad de fabricantes de software de correo de Internet de cliente y de servidor; el *SMTP* para el trabajo del lado del servidor y el *POP3 o IMAP4* en el lado del cliente.

2.2.4 Nombres y alias de los buzones de correo.

Los usuarios especifican recipientes, proporcionando pares de cadenas que identifican el nombre de la máquina destino para el correo y una dirección de buzón en dicha máquina. Los nombres utilizados en estas especificaciones son independientes de otros nombres asignados a las máquinas. Usualmente, la dirección de un buzón es la misma que la identificación del usuario ante el sistema, y el nombre de la máquina destino es el mismo que el nombre de dominio de la máquina.

Si disponemos de un acceso a Internet tendremos nuestra propia dirección de correo electrónico, y un espacio denominado buzón en el área de almacenamiento del servidor que funciona como servidor de correo. Dentro de la red global de Internet, las direcciones tienen una forma muy simple y fácil de recordar:

parte-local@ nombre-de-dominio

Donde *nombre-de-dominio* es el nombre de dominio de un destino de correo al que el correo debe ser entregado y *parte-local* es la dirección de un buzón en la máquina. Por ejemplo la dirección de correo electrónico:

crlssnchz@yahoo.com.mx

crlssnchz es el nombre de usuario, lo que corresponde a un buzón en la máquina; *yahoo.com.mx* es el nombre del servidor que almacenará el correo del usuario.

2.2.5 Buzón de correo electrónico

El buzón de correo electrónico es el espacio físico de almacenamiento en el servidor de correo entrante asignado a algún usuario, para que los mensajes enviados a él sean almacenados. Posteriormente él podrá tener acceso a éstos mediante alguna interfaz gráfica. Para administrar su cuenta de correo electrónico el usuario utiliza la interfaz gráfica que su *ISP* le proporciona.

El espacio físico para almacenar mensajes de correo electrónico está dividido en carpetas(folders), cada usuario puede crear nuevas o eliminar otras que él haya creado.

En la mayoría de los sistemas de correo electrónico existen una serie de carpetas estándar, aparte de las que el usuario cree por su cuenta. Dichas carpetas son:

- Carpeta Bandeja de entrada(también conocida como "Buzón" o *"INBOX"*). Es la carpeta a donde van a parar todos los mensajes, al consultar nuestro correo electrónico.

- Carpeta Bandeja de salida (también denominada *"Unsent messages"*). En esta carpeta se almacenan temporalmente todos los mensajes antes de ser enviados. Esto ocurre cuando se está en modo desconectado.

- Carpeta Borrador (también denominada *"Drafts"*). Si comenzamos a redactar o a responder a un mensaje, pero aún no se desea enviar, sino se quiere guardar para continuar con la redacción más adelante, el mensaje en cuestión se almacenará en esta carpeta.

- Carpeta Elementos eliminados (también denominada *"Papelera" o "Trash"*). Es la carpeta a la que van a parar todos los mensajes eliminados.

3. Protocolos de correo electrónico

El objetivo del conjunto de protocolos *TCP/IP* es proporcionar interoperatividad a través de un amplio rango de sistemas de computadoras y redes. Para extender la interoperatividad del correo electrónico, el *TCP/IP* divide sus estándares de correo en dos grupos. Un estándar especifica el formato para los mensajes de correo, cuyo nombre es ***RFC 822***(contiene un análisis completo de la cabecera). El otro especifica los detalles del intercambio de correo electrónico entre dos computadoras. Mantener los dos estándares separados para el correo electrónico hace posible construir compuertas de correo que conecten redes de redes *TCP/IP* con algunos sistemas de entrega de correo de otros vendedores, siempre y cuando utilicen el mismo formato de mensajes para ambos.

El estándar *TCP/I*P para los mensajes de correo especifica el formato exacto de los encabezados de correo así como el significado de interpretación de cada campo del encabezado; la definición del formato del cuerpo se deja al emisor. En particular, el estándar especifica que los encabezados contienen texto que es posible leer, dividido en líneas que consisten en palabras clave, seguidas por puntos y un valor. Algunas palabras clave son necesarias, otras son opcionales y el resto no tiene interpretación. Por ejemplo, el encabezado debe contener una línea que especifique el destino. La línea comienza con un ***To***: y el resto contiene la dirección del correo electrónico del recipiente proyectado. Una línea que comienza con ***From***: contiene la dirección de correo electrónico del emisor. Opcionalmente, el emisor puede especificar una dirección a la que se pueden enviar réplicas(esto es, para permitir al emisor especificar que las replicas deben enviarse a una dirección diferente al buzón del emisor). Si está presente, una línea que comienza con ***Reply-To***: especifica la dirección para las réplicas. Si esta línea no existe, el recipiente usará la información en la línea ***From***: como la dirección de retorno.

El formato de los mensajes de correo ha sido seleccionado para facilitar el proceso y realizar transporte a través de máquinas heterogéneas. Mantener el formato del encabezado de correo sin cambios permite utilizarlo dentro de un amplio rango de sistemas. La restricción de los mensajes al formato de sólo texto evita los problemas de seleccionar una representación binaria estándar y traducir entre la representación estándar y la representación de la máquina local.

3.1 Protocolo *SMTP* (Simple Mail Transfer Protocol)

Además del formato de los mensajes, el conjunto de protocolos *TCP/IP* especifica un formato estándar para el intercambio de mensajes. Es decir, el estándar especifica el formato exacto de los mensajes a un cliente en una máquina que lo utiliza para transferir correo hacia el servidor en otra. El protocolo de transferencia estándar se conoce como *SMTP*, Simple Mail Transfer Protocol(Protocolo simple de transferencia de correo). Este protocolo se enfoca específicamente en cómo transfiere el sistema de entrega de correo subyacente los mensajes, a través de un enlace de una máquina a otra. No especifica de qué manera acepta el sistema de correo los mensajes de un usuario, o cómo presenta al usuario la interfaz de usuario entrante.

Cuando un cliente *SMTP* tiene un mensaje para transmitir establece un canal bidireccional hacia un servidor *SMTP*. Un servidor *SMTP* puede ser el último destino, un retransmisor intermediario o un *gateway*(el transporte del mensaje se efectúa utilizando un protocolo distinto al *SMTP*, utilizado en sistemas de correo electrónico propietarios). Los comandos *SMTP* son generados por el cliente, y enviados al servidor. Las réplicas son enviadas del servidor *SMTP* hacia el cliente en respuesta a los comandos; la transferencia de un mensaje puede ocurrir en una única conexión entre el remitente original(no un retransmisor intermediario) y el receptor *SMTP* final, o puede ocurrir una serie de saltos a través de sistemas intermediarios. En cualquiera de los casos, se lleva a cabo un trato de responsabilidad para el mensaje: el protocolo requiere que el servidor acepte la responsabilidad para depositar el mensaje, o reportar algún error ocurrido.

Como resultado de una petición de envío de correo por parte del usuario, el emisor *SMTP*, que actúa como cliente, establece una conexión *TCP* con el receptor *SMTP*, que hace la función de servidor. Este último puede ser el equipo final al cual va dirigido el mensaje o bien un sistema intermedio. El puerto definido para llevar a cabo una conexión *SMTP* es el 25.

Una vez que el canal ha sido establecido (comando *HELO*), el emisor envía un comando *MAIL* indicando quien envía el mensaje. Si el receptor *SMTP* puede aceptar correo, responderá con una repuesta *OK*. Posteriormente, el emisor *SMTP* envía un comando *RCPT* (recipient) identificando al destinatario del mensaje. Si el receptor *SMTP* puede aceptar mensajes para dicho destinatario, responderá mediante un *OK*, en caso contrario rechazará el mensaje. Tanto el cliente como el servidor pueden negociar varios destinatarios. Una vez que han llegado a un acuerdo, el cliente envía los datos del mensaje (comando *DATA*). Si el servidor recibe los datos de manera correcta, responderá con el comando *OK*. El canal se cerrará con el comando *QUIT*.

SMTP proporciona diferentes mecanismos para la retransmisión de correo:

- Directamente desde el *host* del usuario emisor hasta el *host* del usuario receptor, cuando ambos *host* están conectados al mismo servicio de transporte.
- A través de uno o más servidores *SMTP* (que retransmiten el mensaje), cuando origen y destino no se encuentran en el mismo servicio de transporte.

Para poder proporcionar la capacidad de retransmisión, el servidor *SMTP* necesita conocer el nombre del *host* destino, así como el nombre del buzón al cual va dirigido el mensaje de correo.

El argumento que admite el comando *MAIL* especifica el emisor del mensaje. También conocido como camino inverso. Este camino se va modificando a medida que el mensaje atraviesa diferentes servidores de correo y se utiliza para obtener el camino de vuelta cuando exista la necesidad de notificar mensajes de error.

El argumento que admite el comando *RCPT* especifica a quién va dirigido el mensaje.

Cuando el mensaje va dirigido a varios destinatarios dentro de un mismo *host*, el servidor *SMTP* únicamente envía una copia de dicho mensaje.

3.1.1 Procedimientos *SMTP*

Inicio de sesión:

Es iniciada una sesión *SMTP* cuando el cliente abre una conexión hacia el servidor, y el servidor le responde con un mensaje de bienvenida, cuyo código es 220. Cuando un servidor rechaza la petición del cliente, le envía un mensaje cuyo código es 554.

Iniciación del cliente:

Una vez que el servidor ha enviado el mensaje de bienvenida, y el cliente lo ha recibido, éste le envía el comando *EHLO* indicando su identidad, de esta manera el cliente también le indica al servidor que está habilitado para el servicio de extensiones, por lo cual solicita al servidor una lista de las extensiones que soporta.

Sistemas que funcionan con anteriores versiones de *SMTP* requieren el comando *HELO* en lugar de *EHLO* por parte del cliente.

Transacciones de correo:

Hay tres pasos para llevar a cabo una transacción de correo:

El primero de ellos se produce al escribir el comando *MAIL*, el cual identifica al usuario emisor.

El esquema es el siguiente:

MAIL FROM:<reverse-path> [SP<mail-parameters>]<CRLF>

La secuencia *<CRLF>* indica la pulsación de la tecla *ENTER*(que produce un retorno de carro y un avance de línea).

Este comando avisa al receptor *SMTP* de que una nueva transacción está comenzando y por lo tanto debe vaciar los *buffers* de memoria con el motivo de dejar espacio libre para el mensaje entrante. En el caso de aceptar el envío del comando, el receptor envía una respuesta 250 *OK*.

El argumento *<reverse-path>* es utilizado para enviar reportes de posibles errores que ocurran. El parámetro *[SP<mail-parameters>]*, es opcional, se utiliza para anexar algunas opciones para los servicios de extensión.

El segundo paso consiste en especificar uno o varios comandos de tipo *RCPT*, los cuales identifican a los usuarios finales a quienes se les quiere hacer llegar el mensaje. Un ejemplo es:

RCPT TO: <forward-path> <[SP<rcpt-parameters>] CRLF>

Este comando identifica el destino al cual va dirigido el mensaje. En caso de ser aceptado, el receptor *SMTP* envía una respuesta 250 *OK*. En el caso de tratarse de un destino desconocido, el receptor *SMTP* envía una respuesta con el identificador 550. El parámetro *<[SP<rcpt-parameters>]* es opcional, se utiliza para establecer datos con los servicios de extensión.

El tercer paso es el envío del mensaje de correo en sí, precedido por la orden *DATA*. El final del mensaje se indica mediante la aparición de una línea cuyo primer carácter es un punto. Si el argumento es aceptado, inmediatamente se envía una respuesta con le código 354. Desde ese momento, y hasta que se encuentre una línea del mensaje que comienza con un punto, el receptor considera que todas las líneas que le llegan forman el mensaje.

3.1.2 Comandos y respuestas

Los comandos y respuestas pueden ir escritos en mayúsculas, minúsculas o una mezcla de ambas. Estos comandos y respuestas se componen de caracteres *ASCII*. Cuando el servicio de transporte proporciona un canal de transmisión de 8 *bits*, cada carácter de 7 bits se envía colocando un cero en el *bit* de mayor peso.

SMTP proporciona características adicionales, tales como verificar el nombre de un usuario o expandir una lista de correo. Para ello se utilizan los comandos *VRFY y EXPN*.

3.1.3 Encaminamiento de los mensajes

El receptor *SMTP* puede ser el destino final del mensaje o bien un sistema intermediario. Esto significa que durante el envío de un mensaje de correo, diferentes *hosts* de la red hacen las funciones de cliente y servidor.

Cuando un *host* recibe un mensaje de correo proveniente de otro equipo, debe verificar si el destinatario es él mismo o si por el contrario debe encaminar el mensaje hacia el siguiente host, indicado en la lista para que éste a su vez lo encamine hasta que llegue a su destino.

Los caminos indicados en *MAIL y RCPT* son diferentes, ya que en el primer caso se trata de la ruta por la cual viaja el mensaje hasta llegar a su destino, y en el segundo es una dirección absoluta que identifica el destinatario del mensaje.

Cuando un *host* recibe un mensaje dirigido a un usuario que pertenece a otro *host*, puede aceptar o rechazar la labor de encaminar el mensaje hasta el otro equipo. En el caso de aceptarla, coloca su propio identificador de *host* al principio de la lista de *hosts* que

aparecen en el comando *MAIL*. En ese momento, el *host* comienza su papel de cliente y establece una conexión *TCP* con el siguiente *host*(que hará la función de servidor).

Cuando un *host* recibe un mensaje y detecta cualquier anomalía en la dirección destino impidiendo así la entrega del correo, debe generar un mensaje de error y enviarlo al emisor utilizando para ello el camino inverso indicado en el comando *MAIL*. Este camino permite identificar a todos los *hosts* por lo cuales ha ido atravesando el mensaje hasta llegar al emisor del mensaje.

Debido a que un receptor de correo *SMTP* es un servidor, y *SMTP* es una aplicación punto-a-punto más que de retransmisión, es necesario que el servidor esté disponible cuando un cliente desea enviarle correo. Si el servidor *SMTP* reside en una estación de trabajo o en un *PC*, ese *host* debe estar ejecutando cuando el cliente quiera transmitir. Esto no suele ser un problema en sistemas multiusuario porque están disponibles la mayor parte del tiempo. En sistemas monousuario, sin embargo, este no es el caso, y se requiere un método para asegurar que el usuario tiene un buzón disponible en otro servidor. La estrategia más simple es, usar un sistema multiusuario para las funciones de correo, pero esto no suele ser deseable.

Un método intermedio es descargar la función de servidor *SMTP* de la estación de trabajo del usuario final, pero no la función de cliente. Es decir, el usuario envía correo directamente desde la estación, pero tiene un buzón en un servidor. El usuario debe conectar con el servidor para recoger su correo.

POP e IMAP describen cómo un programa que se ejecuta en una estación de trabajo final puede recibir correo almacenado en un sistema servidor de correo.

3.2 PROTOCOLO *MIME* (Multipurpose Internet Mail Extensión)

Para permitir la transmisión de datos no *ASCII* a través del correo electrónico, la IETF definió la Multipurpose Internet Mail Extensión*(MIME*). La *MIME* no cambia al *SMTP* ni la reemplaza. La *MIME* permite que datos arbitrarios sigan codificándose en *ASCII* y luego se envíen por medio por medio de mensajes de correo estándar. Para adaptarse a tipos y representaciones arbitrarias de datos, cada mensaje *MIME* incluye datos que informan al recipiente del tipo de datos y de la codificación utilizada. La información de *MIME* reside en el encabezado de correo definido en la *RFC 822*, la línea del encabezado *MIME* especifica la versión utilizada, el tipo de datos que se envía y la codificación empelada para convertir los datos en *ASCII*. La figura 1 ilustra un mensaje *MIME* que contiene una fotografía en la representación estándar *GIF*. La imagen *GIF* ha sido convertida en una representación *ASCII* de siete bits mediante la codificación *base64*.

From: crlssnchz@unam.mx
To: crlssnchz@unam.mx
MIME-Version: 1.0
Content-Type: image/gif
Content-Transfer-Encoding: base64

Figura 3.1

En la figura 3.1, los dos primeros campos son específicos del protocolo *SMTP*, que definen al emisor y al receptor del mensaje; la línea del encabezado *MIME*-Version: establece que el mensaje fue compuesto por medio de la versión 1.0 del protocolo *MIME*. *Content-Type*: establece que la información es una imagen *GIF* y *Content-Transfer-Encoding* especifica el tipo de codificación utilizada; el encabezado establece que la decodificación *base64* se utilizó para convertir la imagen a caracteres *ASCII*. Para ver la imagen, establece que el sistema de correo del receptor debe convertir la codificación base64 a binario, y luego correr una aplicación que presente una imagen *GIF* en la pantalla del usuario.

El estándar *MIME* especifica que una declaración *Content-Type* debe contener dos identificadores: un tipo de contenido y un subtipo, separados por una diagonal. En el ejemplo, imagen es el tipo de contenido y *GIF* es el subtipo.

El estándar define siete tipos de contenidos básicos, los subtipos válidos para cada uno y las codificaciones de transferencia. Por ejemplo, aun cuando una imagen puede tener subtipos *JPG* o *GIF*, el texto no puede utilizar ningún subtipo. Además de los tipos estándar y los subtipos, *MIME* permite a un emisor y a un receptor definir tipos de contenido privado.

Los tipos de contenido *(Content-Type)* posibles son siete:

Content-Type	**Descripción.**
text	Texto. Permite especificar el conjunto de caracteres utilizado
image	Imágenes estáticas o generadas en computadora.
audio	Grabaciones de sonido.
video	Grabaciones de video que incluyen movimiento.
application	Para envío de programas ejecutables.
multipart	Mensajes múltiples de los que cada uno tiene una codificación y un tipo de contenido diferentes.
message	Mensajes de correo electrónico completos o referencias externas.

3.2.1 Mensajes *MIME multipart*

El tipo de contenido *multipart* de *MIME* es útil ya que añade una flexibilidad considerable. El estándar define los siguientes subtipos:

Tipo	Descripción
mixed	Indica que un solo mensaje contenga submensajes independientes, de los que cada uno tiene un tipo independiente y una codificación diferente.
alternative	Indica que un solo mensaje incluya varias representaciones de los mismos datos.
parallel	Indica que el mensaje incluye subpartes que deben ser ejecutadas al mismo tiempo.
digest	Indica que un mensaje tiene un conjunto de mensajes.

Los mensajes *multipart* mezclados*(mixed)* hacen posible incluir textos, gráficos y audio en un solo mensaje, o permiten el envío de un documento con segmentos de dato adicionales asociados.

El tipo de codificación *(Content-Transfer-Encoding)* puede ser:

Tipo	**Descripción.**
Quoted-printable	Utilizado en secuencias alfanuméricas. Permite representar los datos superiores a 127 por una secuencia de caracteres especial "=XX" donde XX es el valor hexadecimal del carácter a representar.
Base64	Para representar secuencias de bytes. Permite representar 3 bytes con 4 caracteres ASCII con 6 bits significativos.

3.2.2 Codificación *Quoted-printable*

Un ejemplo de esta codificación es el siguiente:

Texto: ALEX
Codificación: *=41=4C=45=58*

El problema de este tipo de codificación es que para el envío de un simple carácter *ASCII* es necesario llevar a cabo el envío de tres caracteres. En el caso de la letra A, su codificación hexadecimal es *0x41*, pero en su envío es necesario codificar los caracteres *=,4,1*; por lo que en realidad, se triplica el envío de los datos.

Sin embargo, este tipo de codificación nos permite el envío de caracteres *ASCII* que no son habituales en el lenguaje cotidiano. Mezclando caracteres *ASCII* con la codificación *Quoted-printable* es posible enviar el siguiente texto:

Texto: Mª Zurro
Codificación: *M=A6Zurro*

El programa agente de usuario que se está ejecutando en la máquina destino será el encargado de interpretar la extensión *MIME* y traducir este tipo de codificación.

3.2.3 Codificación *base64*

Este tipo de codificación incrementa el tamaño de los mensajes en tan solo una tercera parte. Una vez llevada a cabo la codificación, los programas envían un conjuntos de 24 *bits* como 4 grupos de datos de 6 *bits* cada uno. Para ello utilizan una tabla de codificación -figura 2- . El valor que se encuentra en la parte izquierda de la celda es la codificación en 6 bits del carácter de la parte de la derecha.

Cuando los datos no son múltiplos de un bloque de datos de 3 *bytes*, se utiliza el signo igual *"="* para completar hasta obtener un múltiplo de 3 *bytes.*
Ejemplo:

La palabra ALEX tiene cuatro caracteres, es decir necesita 2 caracteres extras hasta obtener un múltiplo de 3, por lo que se añaden dos caracteres *"="*

Palabra inicial: ALEX
Palabra a codificar: *ALEX*==

Hexadecimal*:*	*0x41*	*0x4C*	*0x45*	*0x58*
Binario:	01000001	01001100	01000101	01011000

Tomando grupos de seis bits codificamos de la siguiente forma:

Codificación*:*	*010000*	*010100*	*110001*	*000101*	*010110*	*000000*
Decimal:	16	20	49	5	22	0
Base64:	Q	U	x	F	W	A

Es importante tener en cuenta que para formar el segundo conjunto de 6 *bits* se han tomado 2 bits del carácter 'A' y 4 *bits* del carácter 'L'. En el caso del último conjunto de 6 *bits*, se han tomado los 2 últimos bits del carácter 'X' y se han añadido cuatro ceros para poder formar otro grupo de 6 *bits*(2 ceros por cada signo '=' añadido).

3.3 *POP3* (POST OFFICE PROTOCOL Versión 3)

El protocolo *POP*(Protocolo de Oficina de Correo) versión 3 es utilizado para transferir correo desde un servidor hasta una estación de trabajo.

El servicio *POP3* se encuentra disponible de forma estándar en el puerto *TCP* 110. Cuando un cliente desea utilizar dicho servicio, establece una conexión *TCP* con el servidor. Una vez que la conexión ha sido realizada, tanto servidor como cliente comienzan un diálogo.

El cliente y el servidor *POP3* intercambian una serie de comandos y respuestas hasta que la conexión finaliza de manera ordenada, o la conexión es abortada por alguno de ellos. Los comandos del protocolo *POP3* consisten de una palabra reservada seguida de uno o varios argumentos. Todos los comandos finalizan mediante la secuencia de caracteres <*CRLF*> (retorno de carro línea nueva) y constan de tres o cuatro caracteres *ASCII*. Los argumentos, pueden tener hasta un máximo de 40 caracteres.

El servidor por su parte envía una serie de respuestas a los comandos. Estas respuestas pueden ser afirmativas *(+OK)* o negativas *(-ERR)*, y pueden ocupar una o más líneas; en el caso de ser multilínea, cada una de éstas va finalizada con la secuencia <*CRLF*>.

3.3.1 Estados del protocolo *POP3*

Cuando se trabaja con el protocolo *POP3* durante la interacción, el cliente es situado en tres distintos estados: autorización, transacción y actualización, de los cuales dependen las operaciones que podrá llevar a cabo.

3.3.1.1 Estado de autorización

Una vez que el cliente ha iniciado una conexión *TCP* al puerto 110, el servidor *POP3* le responde con un mensaje de bienvenida. En este momento la sesión ha entrado en el estado de autorización. El cliente debe identificarse ante el servidor enviando en primer lugar su nombre de usuario y a continuación la contraseña.

El nombre de usuario va precedido del comando *USER* y la contraseña va precedida del comando *PASS*.

USER crls <CRLF>
+OK
PASS crlsss <CRLF>

Cuando el cliente envía el comando *USER* seguido del nombre de usuario, el servidor responderá de una forma u otra dependiendo de la validez de dicho nombre. En el caso de que el nombre de usuario sea correcto, el servidor enviará la cadena *+OK*, y el cliente procederá al envío de la contraseña (comando *PASS*) o bien puede cerrar la conexión mediante el comando *QUIT*. Si el nombre del usuario es incorrecto, el servidor enviará la

cadena –ERR y el cliente tendrá que volver a enviar el comando *USER* o bien finalizar la conexión.

De la misma forma que el resto de los protocolos *TCP/IP*, la información que se envía por la red viaja tal y como se ha escrito(texto *ASCII*), quedando expuesta a cualquier persona que pueda obtenerla. Existe un comando opcional, denominado *APOP*, que permite identificar a un usuario evitando los problemas de seguridad de *USER* y *PASS*.

Todo servidor *POP3* que implemente el comando *APOP*, incluye un sello de tiempo en el saludo inicial. Este sello de tiempo tiene la forma siguiente:

process-id.clock@host

donde *process-id* es el *PID* del proceso, *clock* es la hora del sistema y *host* es el nombre del dominio en le cual se está ejecutando el servidor *POP3*. El cliente *POP3* utiliza este sello de tiempo y ejecuta el comando *APOP* pasando dos argumentos:

APOP nombre de cadena**.**

El argumento nombre es el nombre del usuario. El argumento cadena se calcula aplicando el algoritmo *MD5* a una cadena que está formada por el sello de tiempo y una firma única conocida por el cliente y el servidor. Esta cadena tiene un tamaño de 16 *bytes* y es enviada en formato hexadecimal utilizando letras minúsculas.

Cuando el servidor *POP3* recibe el comando *APOP* verifica la cadena enviada. En caso de que ésta sea correcta, el servidor responde de manera afirmativa y se pasa al estado de transacción. En caso contrario, se envía una respuesta negativa y la conexión permanece en el estado actual. Este método de autentificación es más seguro, ya que la contraseña no viaja desprotegida por la red. Aunque la cadena *MD5* sea capturada por alguien, será muy difícil obtener la información original a partir de la cual se generó.

3.3.1.2 Estado de Transacción

Una vez que el cliente se ha identificado satisfactoriamente ante el servidor *POP3* y éste ha abierto el buzón correspondiente, el cliente puede enviar cualquiera de los comandos válidos en este estado de sesión.

Comando.	Descripción.
STAT	Solicita el estado del buzón. El servidor responde afirmativamente y devuelve el número de mensajes que hay en el buzón y el tamaño que ocupan.
LIST	Se utiliza para solicitar la estadística del buzón. Se puede invocar pasando como argumento el número del mensaje para el cual se desea ver la estadística.
RETR	Permite recuperar un mensaje del buzón siempre que éste no esté marcado para borrar. El comando se invoca pasando como argumento un número de mensaje.
DELE	Marca un mensaje para eliminar. La eliminación se produce cuando se entra en la fase de actualización.
NOOP	El servidor *POP3* no hace nada ante este comando, simplemente responde afirmativamente. Permite mantener activa la conexión *TCP* cuando no hay actividad por parte del cliente.
RSET	Elimina las marcas de los mensajes marcados para borrar mediante el comando *DELE*.
TOP	Permite recuperar cierto número de líneas de un mensaje que se le indica como argumento.
UIDL	Solicita el identificador único de los mensajes almacenados. Este identificador es una cadena que identifica cada mensaje del servidor.

En este estado el servidor prepara toda la información relativa al cliente, para que éste pueda manipular su buzón de correo mediante una serie de comandos específicos. Tras la ejecución del comando *QUIT*, la sesión entra en un estado denominado estado de actualización.

Un servidor *POP3* pude tener un periodo de inactividad de hasta 10 minutos, momento en el cual se cierra la conexión.
Ejemplo de diálogo en la fase de transacción:

STAT
+ OK 3 10320
El usuario ha consultado el buzón y el servidor le informa de que tiene tres mensajes, los cuales ocupan 10320 *bytes*.

Es posible obtener información más detallada mediante el comando *LIST*

LIST 1
+OK 1 100

De esta manera el servidor informa que el mensaje número 1 ocupa 100 *bytes*.

3.3.1.3 Estado de actualización

La llegada a este estado se produce tras la ejecución del comando *QUIT*. En esta fase, el buzón de correo del usuario se desbloquea y se eliminan todos aquellos mensajes que fueron marcados para borrar. Las respuestas posibles que devuelve el servidor son:

+ OK nombre_servidor POP3 server signing off
+ ERR some deleted messages not removed

3.4 Protocolo *IMAP* (Internet Messsage Acces Protocol).

El protocolo de acceso a mensajes de Internet, versión 4 (*IMAP4*) permite a un cliente acceder y manipular los mensajes de correo electrónico contenidos en un servidor. *IMAP4* permite manipular las carpetas remotas contenedoras de mensajes, llamadas comúnmente "buzones"; ofrece además, operaciones para crear, borrar, y renombrar estos buzones; comprobación de nuevos mensajes; borrado permanente de mensajes; establecimiento y borrado de banderas; procesado *RFC-822*; búsqueda genérica; y recuperación selectiva de atributos de mensaje, textos, y porciones de texto.

El acceso a mensajes mediante *IMAP4* se realiza a través de números. Estos números pueden ser, una secuencia de números, o bien identificadores únicos.

El protocolo *IMAP4* confía en la existencia de un flujo de datos estable, como los suministrados por *TCP*. Cuando se utiliza *TCP*, un servidor *IMAP4* escucha el puerto 143.

3.4.1 Transmisor de protocolo cliente y Receptor de protocolo servidor

El comando de cliente da comienzo a una operación. A cada comando de cliente se le asigna un prefijo consistente en un identificador (normalmente una cadena corta de caracteres alfanuméricos, por ejemplo: *A0001, A0002*, ...) llamado "etiqueta". El cliente asigna a cada comando una etiqueta diferente.

Hay dos casos, en los cuales una línea de cliente no representa un comando completo. En primer lugar, un argumento del comando está dotado de un número de *bytes*; en segundo lugar, los argumentos del comando requieren una realimentación por parte del servidor. En ambos casos, el servidor envía una respuesta de solicitud de continuación del comando si este está preparado para recibir los *bytes* (si es lo adecuado) y el recordatorio del comando. Esta respuesta viene precedida del *token"+"*. El receptor de protocolo de un servidor *IMAP4* lee una línea de comando desde el cliente, analiza el comando y sus argumentos, y transmite los datos del servidor y una respuesta de transacción completa de comando.

3.4.2 Transmisor de protocolo servidor y Receptor de protocolo cliente

Los datos transmitidos por el servidor hacia el cliente y las respuestas de estado que no indiquen el procesado completo del comando van precedidas del *token "*"*, y se denominan respuestas no etiquetadas.

La respuesta de transacción completa por parte del servidor indica el éxito o fracaso de la operación. Esta es etiquetada con la misma etiqueta con la que el comando del cliente

comenzó la operación. Por tanto, si hay más de un comando en progreso, la etiqueta de una respuesta de transacción completa de un servidor identifica al comando al cual corresponde la respuesta. Hay tres posibles respuestas de transacción completa por parte del servidor: *OK* (indica éxito), *NO* (indica fracaso), o *BAD* (indica un error de protocolo, tal como comando no reconocido, o error de sintaxis para el comando).

3.4.3 Atributos del Mensaje

Además del texto del mensaje, cada uno de los mensajes posee una serie de atributos. Estos atributos se pueden recuperar individualmente o en conjunción con otros atributos o textos del mensaje.

3.4.3.1 Número de secuencia de mensaje

Posición relativa desde 1 al número de mensajes en el buzón.
Esta posición debe estar ordenada ascendentemente por identificador único. Conforme se añadan nuevos mensajes, se le asignará a cada uno un número de secuencia de mensaje que sea una unidad mayor que el número total de mensajes presentes previamente en el buzón.

Los números de secuencia de mensaje pueden ser reasignados a lo largo de la sesión. Además de tener acceso a los mensajes por su posición relativa en el buzón,

3.4.3.2 Atributo de mensaje Banderas

Las banderas son una lista de cero o más *tokens* dotados de nombre asociados con el mensaje. Una bandera se establece añadiéndola a dicha lista, y se elimina borrándola de la misma. Una bandera puede ser permanente o de sesión única. Una bandera de sistema es un nombre de bandera que viene predefinida en la especificación del protocolo *IMAP*. Todas las banderas de sistema vienen precedidas de "\". Las banderas de sistema definidas actualmente son:

Bandera	Descripción
Seen	El mensaje ha sido leído
Answered	El mensaje ha sido respondido
Flagged	El mensaje está "reseñado" por merecer una atención especial o urgente
Deleted	El mensaje está "borrado" para ser eliminado posteriormente mediante la operación *EXPUNGE*.
Draft	El mensaje está inacabado.
Recent	El mensaje ha llegado recientemente al buzón. Esta sesión es la primera en la que se notificará la llegada de este mensaje, en sesiones posteriores este flag será deshabilitado para este mensaje en concreto. Este flag no puede ser modificado por el cliente.

Una bandera puede ser permanente o de sesión única partiendo de una base por-bandera.

Las banderas permanentes son aquellas que el cliente puede añadir o eliminar de las banderas del mensaje de forma permanente; es decir, las sesiones posteriores serán conscientes de cualquier cambio en estas banderas permanentes. Los cambios realizados sobre las banderas de sesión sólo serán válidos en esa sesión en concreto.

3.4.3.3 Atributo de mensaje fecha interna

Esta no es la fecha y hora de la cabecera *RFC-822*, sino que es una fecha y una hora que refleja el momento de recepción del mensaje. En el caso de mensajes distribuidos vía *SMTP*, es la fecha y hora de distribución final del mensaje como se define por *SMTP*. En el caso de los mensajes enviados mediante el comando *COPY* de *IMAP4*, esta debe ser la fecha y hora interna del mensaje origen. En el caso de ser enviado mediante el comando *APPEND* de *IMAP1*, esta debe ser la fecha y hora especificada en la descripción del comando *APPEND*.

3.4.4 Diagrama de estado.

Un servidor *IMAP4* puede encontrarse en cuatro estados. La mayoría de los comandos son válidos únicamente en ciertos estados. Es un error de protocolo de parte del cliente intentar ejecutar un comando mientras este no se encuentra en el estado apropiado. En este caso, un servidor responderá con un resultado de fin de procesado de comando *BAD o NO* (dependiendo de la implementación del servidor).

3.4.4.1 Estado no autenticado

En este estado, el cliente debe suministrar credenciales de autentificación antes de que la mayoría de los comandos puedan ser aceptados. Este estado se alcanza cuando comienza la conexión salvo que la conexión haya sido pre-autenticada.

En el estado no autentificado, el comando *AUTHENTICATE o LOGIN* establece la autentificación y nos introduce en un estado autentificado. El comando *AUTHENTICATE* ofrece un mecanismo global para una gran variedad de técnicas de autentificación, mientras que el comando *LOGIN* hace uso del par nombre de usuario tradicional y contraseña de texto plano.

3.4.4.2 Estado autenticado

En este estado, el cliente está autentificado y debe seleccionar un buzón antes de que se le permita utilizar cualquier comando que afecte a los mensajes contenidos en el mismo. Este estado se alcanza cuando comienza una conexión pre-autenticada, cuando se suministran credenciales de autentificación válidas, o tras un error debido a la selección de un buzón.

En el estado autentificado, se permite utilizar los comandos que manipulan el buzón de correo. Los comandos *SELECT, EXAMINE, CREATE, DELETE, RENAME, SUBSCRIBE, UNSUBSCRIBE, LIST, LSUB, STATUS, y APPEND* son válidos en este estado.

Los comandos *SELECT y EXAMINE* seleccionarán un buzón de correo para conseguir acceso y entrada al estado seleccionado.

3.4.4.3 Estado seleccionado

En este estado, se ha seleccionado un buzón al que acceder. Este estado se alcanza cuando la selección del buzón es aceptada.

En el estado seleccionado, están permitidos los comandos que manipulan mensajes en un buzón. Además de los comandos de estado autentificado son válidos los siguientes en el estado seleccionado: *CHECK, CLOSE, EXPUNGE, SEARCH, FETCH, STORE, COPY, y UID.*

3.4.4.4 Estado desconexión.

En este estado, la conexión está en proceso de finalización, y el servidor cerrará la conexión. Este estado puede alcanzarse como resultado de la petición del cliente o por decisión unilateral del servidor.

Los siguientes comandos son perfectamente válidos en cualquiera de los estados: *CAPABILITY, NOOP, y LOGOUT*.

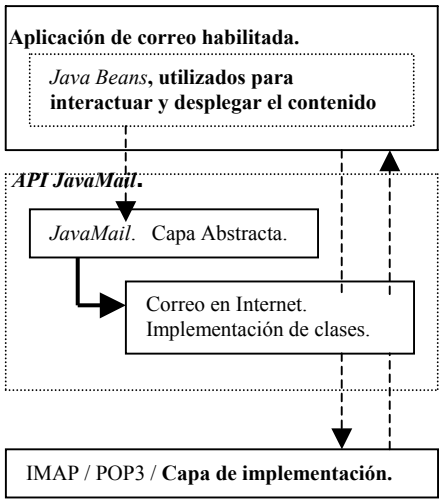
El comando *CAPABILITY* solicita un listado de todas las aptitudes de que dispone el servidor. Este listado de aptitudes no depende del estado de la conexión o del usuario. Por lo tanto, no es necesario elaborar un comando *CAPABILITY* más de una vez por conexión.

El comando *NOOP* efectúa ninguna tarea. Puesto que un comando puede devolver como dato no etiquetado una actualización de estado, el comando *NOOP* puede ser utilizado para comprobación de nuevos mensajes o de actualizaciones sobre el estado de los mensajes durante un periodo de inactividad. El comando *NOOP* también se puede utilizar para inicializar cualquier contador de desconexión por inactividad presente en el servidor**.**

El comando *LOGOUT* informa al servidor de que el cliente ha conseguido conectarse. El servidor debe enviar una respuesta no etiquetada *BYE* antes de la respuesta (etiquetada) *OK*, y a continuación cerrar la conexión de red.

4. *API JavaMail*

JavaMail provee elementos utilizados para desarrollar interfaces de sistemas de mensajes, no proporciona una implementación en particular, ya que define diversas clases que implementan los protocolos estándar*(RFC 822 y MIME*) para envío de correo electrónico a través de Internet, con el objetivo de ejecutar funciones que abarcan el manejo de los procesos estándar para una aplicación cliente, algunas de ellas son: envío, recuperación y eliminación de correo electrónico.



Esquema 4.1 Arquitectura *JavaMail.*

La arquitectura de *JavaMail* está integrada por tres capas:

- Capa abstracta: se declaran interfaces, clases abstractas y métodos con el objetivo de dar soporte a las funciones de los sistemas de correo electrónico basados en estándares abiertos.

- Capa de implementación: implementa la capa abstracta utilizando los protocolos de estándares abiertos para correo electrónico en Internet(*RFC822 y MIME*).

- Capa *JAF (Java Beans Activation FrameWork*): *JavaMail* debe utilizar los *JAF* ya que, a través de los *Beans* se manipula todo el contenido del mensaje del correo electrónico. Los *JAF* no son parte de la *API JavaMail*.

Administrar nuestro correo electrónico es muy simple para quienes utilizamos alguna interfaz de usuario(*Outlook, hotmail, terra, prodigy, yahoo*,.....) por que únicamente seleccionamos opciones de los diversos menús(enviar, leer, redactar ....); mediante la *API JavaMail* es posible desarrollar este tipo de interfaz de usuario, ya que proporciona diversas clases utilizadas para implementar protocolos de estándares abiertos de correo electrónico, de esta manera podemos automatizar su administración(responder mensajes con base en el

asunto, eliminarlos cuando el remitente sea desconocido, ejecutar algún proceso cuando llegue el mensaje ....).

La *API JavaMail* añade el concepto de registro de proveedores, lo cual simplemente es la definición de un conjunto de especificaciones para anexar a la *API* nuevas clases que implementan versiones nuevas o actualizadas de protocolos, tanto de envío como de obtención de correo electrónico. En la documentación de la *API JavaMail* se anexa la descripción del registro de proveedores.

4.1 Clases más importantes de la *API JavaMail*

4.1.1 La clase *Message*

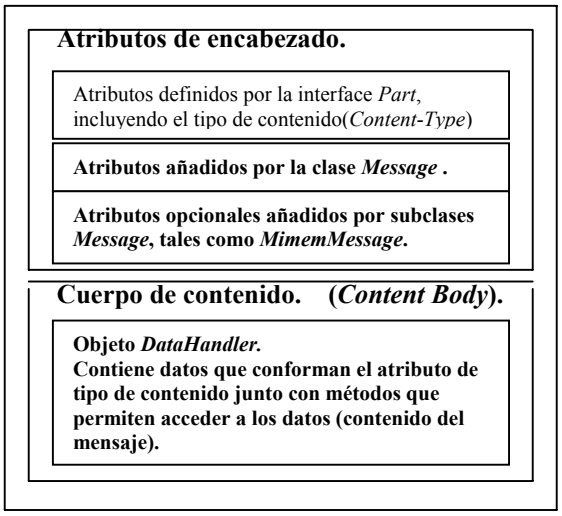
La clase *Message* define un conjunto de atributos y el contenido para los mensajes de correo electrónico. Los atributos definidos por la clase *Message* especifican información de la dirección y definen la estructura del contenido incluyendo tipo de contenido. El contenido está representado por un objeto *DataHandler*.

La clase *Message* es una clase abstracta que implementa la interfaz *Part*; añade atributos que conforman el encabezado de un mensaje de correo electrónico; *From*(Para), *To*(de),Subject(asunto), y *BCC*(con copia oculta para) principalmente.

La *API JavaMail* no conoce el tipo del dato o el formato del contenido del mensaje. Un objeto *Message* interactúa con su contenido a través de una capa intermedia, los *Java Beans Activation Framework*. Esta separación le permite al objeto *Message* manejar cualquier contenido del mensaje y transmitirlo mediante los protocolos estándar para envío de correo electrónico implementados en la *API JavaMail.*

Subclases de la clase *Message* pueden implementar diversos formatos estándar para mensajes de correo electrónico. Por ejemplo, la *API JavaMail* provee la clase *MimeMessage* (subclase de la clase *Message*) que hereda de la clase *Message* para implementar los estándares *RFC822 y MIME*. La subclase *Message* instancia un objeto que posee el contenido del mensaje junto con los atributos que especifican direcciones y recipientes para enviarlo, así como información de la estructura del mensaje y el tipo de contenido. Los mensajes son colocados dentro de una carpeta, y cada mensaje tiene asociado un conjunto de banderas que describen su estado dentro de ésta.

Esquema de la clase *Message*.



Los objetos *Message* son recuperados de los objetos *Folder* o creados haciendo instancias de alguna subclase *Message*. Los mensajes almacenados dentro de un objeto *Folder* están numerados en forma secuencial iniciando en uno. El número asignado cambia cuando un mensaje es eliminado.

Un objeto *Message* puede contener múltiples partes(puede tener muchos *BodyPart*), donde cada parte contiene su propio conjunto de atributos(atributos no referidos a la especificación de direcciones de envío) y contenido. El contenido de un mensaje multiparte es un objeto *Multipart* que contiene objetos *BodyPart* representando cada parte individual.

La interface *Part* define un conjunto estándar de encabezados comunes para muchos sistemas de correo; especifica el tipo de dato del contenido y define un conjunto de métodos para obtener y especificar cada miembro de los encabezados.

El atributo tipo de contenido(*Content-Type*)

El atributo tipo de contenido especifica el tipo de contenido del dato siguiendo la especificación *RFC 2045*. Un tipo *MIME* está compuesto de un tipo primario que declara el tipo general del contenido, y un subtipo que especifica el formato del contenido.

Los componentes de la *API JavaMail* pueden acceder al contenido a través de estos mecanismos:

- Como un flujo de entrada: La interface *Part* declara un método (*getInputStream*) que devuelve un flujo de entrada para acceder al contenido. Es importante destacar que las implementaciones de la interface *Part* deben decodificar cualquier contenido que esté codificado antes de proveer el flujo de entrada.

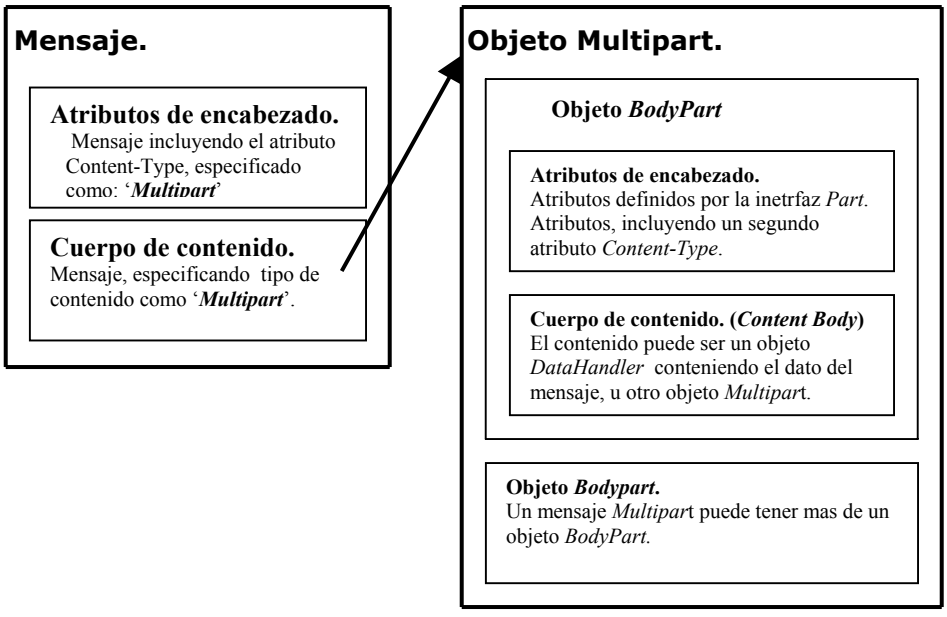
- Como un objeto *DataHandler*: La interface *Part* declara un método (*getDataHandler*) que devuelve un objeto *DataHandler*, el cual contiene los datos(contenido del mensaje); este objeto permite (a los clientes) determinar las operaciones disponibles que pueden ser ejecutarlas sobre el contenido, e instanciar(invocar el *Bean* asociado con el comando elegido) el componente adecuado para ello.

- Como un objeto en el lenguaje de programación Java: La interface *Part* declara un método (*getContent*) que devuelve el contenido como un objeto en el lenguaje de programación Java. El tipo del objeto devuelto depende del tipo del contenido. Si el contenido es de tipo *multipart,* el método *getContent* devolverá un objeto *Multipart*, o un objeto de alguna subclase *Multipart*; este método devuelve un flujo de entrada para tipos de dato desconocidos.

4.1.2 La clase *Multipart*

La clase *Mulitpart* implementa mensajes con varias partes(multipartes). Un mensaje multiparte es un objeto cuyo atributo tipo de contenido (*Contet-Type*) se especifica como *'multipart'*, y el *Content-Body* trae una referecia a un objeto *Multipart*, el cual es un contenedor de objetos *BodyPart*; cada *BodyPart* puede contener un objeto *DataHandler*(objeto con el mensaje) u otro objeto *Multipart*.

Mediante esta clase podemos crear mensajes de correo electrónico cuyo contenido se estructura en distintas secciones, donde cada una de estas tiene asociado un conjunto de especificaciones para determinar el tipo de contenido(texto, imágenes, ...). El esquema de esta clase es el siguiente:



Generalmente el cliente determina (llamando al método *getContent-Type*) el tipo de contenido del mensaje, si obtiene un tipo *MIME* cuyo tipo primario indica que es un mensaje multiparte (*multipart),* entonces debe acceder a cada sección (invocando al método *getContent*, para obtener el contenedor de todas las secciones del mensaje). El objeto *Multipart*, soporta varios métodos para obtener, crear y borrar objetos *BodyPart*.

4.1.3 La clase *Flags*

Cada mensaje almacenado tiene asociado un conjunto de banderas(indicadores) que especifican su estado; nuevo, contestado, eliminado y leído. El objeto *flags* contiene las banderas que describen el estado del objeto mensaje dentro de su *folder*.

Algunas de las banderas son:

ANSWERED: El cliente especifica esta bandera cuando el mensaje ha sido contestado.

RECENT: Esta bandera se especifica cuando el mensaje es nuevo en en el fólder. El cliente no puede especificar el valor de esta bandera. Cuando el mensaje ha sido leído, automática - mente cambia de valor(cambia a FALSE).

SEEN: Marca el mensaje indicando que ya ha sido leído.

DELETED: Indica que el mensaje será eliminado. Sin embargo, para eliminarlo físicamente se debe utilizar otro método después de que esta bandera halla sido especificada a TRUE.

Los posibles valores para cada bandera corresponde a FALSE y TRUE.

4.2 La sesión de correo

La *API JavaMail* provee el objeto *Session*, el cual administra las opciones de configuración y utiliza los datos de autorización(nombre de usuario y contraseña) para interactuar con los sistemas de mensajes.

El objeto *Properties* (propiedades) que inicializa la sesión contiene valores por defecto e información de configuración(servidor de correo, nombre de usuario y contraseña entro otros).

El cliente utiliza métodos para especificar valores por defecto en las propiedades, especialmente el protocolo para acceder a los mensajes (propiedad llamada: *mail.store.protocol*), protocolo de transporte de mensajes (propiedad llamada: *mail.transport.protocol*), nombre del servidor de correo (*mail.host*) y nombre de usuario asignado para establecer la conexión al servidor de correo (*mail.user*).

Algunas implementaciones de sistemas de mensajes pueden utilizar propiedades adicionales. Si la especificación de requerimientos es incompleta, entonces la aplicación puede utilizar el objeto propiedades de sistema. Este objeto es recuperado a través del método *System-getProperties*.

JavaMail soporta simultáneamente múltiples sesiones; en cada una puede acceder a los diversos almacenamientos de mensajes. Cualquier aplicación que necesita acceder en el almacenamiento primario de mensajes puede compartir la sesión por defecto. El correo habilita aplicaciones establecidas en la sesión por defecto, las cuales inicializan la información de autorización necesaria para acceder al fólder *'Inbox'* del usuario. Otras aplicaciones utilizan la sesión por defecto cuando envían o acceden al correo a nombre del usuario. Cuando comparten el objeto sesión todas las aplicaciones, también comparten la información de autorización y propiedades.

El objeto *Authenticator*(autorización) controla aspectos de seguridad para el objeto *Session*. Los sistemas de mensajes utilizan un mecanismo para interactuar con el usuario cuando una contraseña es requerida por el sistema. Los clientes de la *API JavaMail* pueden registrar objetos *PasswordAuthentication* con el objeto *Session* para utilizarlo en sesiones posteriores; debido a que éstos objetos contienen contraseñas su acceso debe ser restringido.

Un cliente de correo que iniciado sesión utiliza el objeto *Session* para recuperar tanto un objeto *Store* (almacenamiento) como un objeto *Transport*(trasnporte) para leer y enviar correos respectivamente; los recupera por que están especificados en las propiedades para dicha sesión.

El cliente puede pasar por alto los parámetros de la sesión por defecto y acceder al objeto *Store* o *Transport* para especificar los protocolos que considere pertinente.

4.2.1 Registro de Proveedores

Registro de proveedores permite a los clientes registrar implementaciones propias de nuevos protocolos para ser utilizados por la *API JavaMail*. Para especificar nuevos protocolos se utilizan los siguientes archivos(se les denomina archivos de recurso).

- *javamail.providers y javamail.default.providers*
- *javamail.address.map y javamail.default.address.map*

Cada archivo de recurso(X) es buscado en la siguiente ruta(para la versión 1.4 del *JDK*):

Para el archivo javamail.default.providers:

- 'directorio_java_home'\jre\lib\ext\mail.jar(javamail.default.providers)

Para el archivo javamail.default.address.map:

- ‘directorio_java_home’\jre\lib\ext\mail.jar(javamail.default.address.map)

El archivo de recurso *javamail.providers* especifica los protocolos para transportar y almacenar mensajes de correo electrónico disponibles en el sistema. El contenido del archivo de recurso es:

 # JavaMail IMAP provider Sun Microsystems, Inc
protocol=imap; type=store; class=com.sun.mail.imap.IMAPStore; vendor=Sun Microsystems, Inc;

 # JavaMail SMTP provider Sun Microsystems, Inc
protocol=smtp; type=transport; class=com.sun.mail.smtp.SMTPTransport; vendor=Sun Microsystems, Inc;

 # JavaMail POP3 provider Sun Microsystems, Inc
protocol=pop3; type=store; class=com.sun.mail.pop3.POP3Store; vendor=Sun Microsystems, Inc;

El archivo de recurso *javamail.address.map* especifica la representación de cada mensaje que se envía mediante el protocolo de transporte utilizado. El formato del archivo es una serie de parejas de nombre-valor, donde nombre indica la representación a utilizarse para el mensaje, y valor indica el protocolo utilizado. El contenido del archivo recurso es:

rfc822=smtp.

4.2.2 Selección de protocolo y valores por defecto

Cuando es creado el objeto sesión(*Session*) inicia la asignación de variables de los archivos de recurso. El orden de los protocolos en el archivo de recurso determina el valor inicial por defecto para las propiedades mail.store.protocol y mail.transport.protocol.

A través de métodos (*getProviders(),getProvider() y setProvider()*) es posible determinar los protocolos disponibles(instalados) y especificar cual será utilizado por defecto.

Una aplicación en tiempo real puede especificar la implementación por defecto para algún protocolo en particular, para ello se utiliza la propiedad *mail.protocol*.

El código puede llamar al método *setProviders()* asignándole un proveedor que fue devuelto por el método *getProviders*(). Un objeto proveedor (*Provider()*) no puede ser creado explícitamente; debe ser recuperado utilizando el método *getProviders()*. En otro caso, el proveedor debe ser especificado con alguna de las opciones configuradas en los archivos de recurso.

Es importante mencionar que los métodos descritos permiten al cliente seleccionar las implementaciones preconfiguradas, pero no es posible configurar una nueva implementación.

4.3 Almacenamiento y recuperación de mensajes

Esta sección describe las facilidades del almacenamiento de mensajes soportadas por las clases *Store*(almacenamiento) y *Folder*(fólder o carpeta). Los mensajes están contenidos en fólderes. Nuevos mensajes son entregados a los fólderes a través de un protocolo de transporte. Los clientes recuperan mensajes de los fólderes utilizando el protocolo de acceso.

4.3.1 La clase almacenamiento(*Store*)

La clase almacenamiento*(Store)* define una base de datos que posee una estructura jerárquica para las carpetas y los mensajes contenidos en éstas. Esta clase también define el protocolo de acceso utilizado para acceder a las carpetas y recuperar los mensajes.

La clase *Store* (almacenamiento) es una clase abstracta, subclases de esta implementación especifican la base de datos de mensajes y protocolos de acceso.

Los clientes logran acceder al almacenamiento de mensajes (*Message Store*) mediante la obtención de un objeto *Store*(almacén) que implementa el protocolo de acceso a la base de datos. Muchos requieren que el usuario sea autorizado antes de permitirle el acceso, el método *connect* (conexión) ejecuta esta autorización. Para algunos almacenadores de mensajes el nombre del *host*, nombre de usuario y la contraseña son suficientes para autorizar al usuario.

La *API* provee el método *connect*(conexión) que toma esta información como parámetros de entrada. La clase almacén(*Store*) provee el método *connect* por defecto. En este caso el cliente espera obtener información del objeto sesión(*Session*) o interactuar en la sesión a través del objeto autorizador(*Authenticator*).

La implementación por defecto para el método connect(conexión) en la clase almacén(Store) utiliza estas técnicas para recuperar toda la información necesaria y entonces llamar al método *protocolConnect*(para acceder al mensaje).

Por defecto *Store* consulta las siguientes propiedades para el nombre de usuario y el nombre del *host*:

- propiedad *mail.user* o propiedad *user.name*
- *mail.host*

Estas propiedades globales pueden ser anuladas sobre un protocolo por las propiedades fundamentales:

- mail.protocol.user
- mail.protocol.host

Los clientes terminan una sesión mediante una llamada al método *close()* sobre el objeto *Store*. Cuando *Store* ha sido cerrado (ya sea explícitamente utilizando el método *close*, o externamente cuando la conexión al servidor de correo falla) todos los componentes de mensajes pertenecientes para este *Store* son inválidos.

4.3.2 Eventos de la clase *Store*

Store envía los siguientes eventos, los cuales pueden ser recibidos por métodos para ejecutar alguna tarea específica:

- *Connection Event*: generado cuando una conexión hecha para el *Store* se efectúa satisfactoriamente, o cuando una conexión existente es terminada o desconectada.

- *Store Event*: envía comunicados de alerta y notificación de mensajes del *Store* para el usuario final. El método *getMessageType()* devuelve el tipo de evento, el cual puede ser de alerta(*ALERT*) o de noticia (*NOTICE*).

- *FolderEvent*: comunica cambios para cualquier fólder contenido dentro del *Store*. Estos cambios incluyen tanto creación como, supresión de un fólder existente y cuando ha sido renombrado alguno.

4.3.3 La clase *Folder*

La clase abstracta *Folder* representa un contenedor para mensajes. Los objetos *Folder* pueden contener subcarpetas y mensajes, de esta manera proveen una estructura jerárquica.

Métodos principales de la clase *Folder.*

getType()	determina si un objeto *Folder* contiene subcarpetas, mensajes o ambos.
getDefaultFolder()	para el correspondiente objeto *Store,* devuelve el *Folder* raíz de la jerarquía por defecto.
list()	devuelve todos las subcarpetas contenidos en le objeto *folder.*
create()	creación de un nuevo fólder en el almacenamiento**.**

Un objeto *Folder* permite diversas operaciones, incluyendo borrar, renombrar, listar subcarpetas y monitorear para nuevos mensajes. El método *open()* abre un objeto fólder. Todos los métodos aplicables sobre objetos *Folder* excepto *open, delete y renameTo* son válidos cuando están en estado de abierto.

Los mensajes dentro de un fólder son numerados a partir de uno, la numeración se basa de acuerdo a la hora en que son depositados. Cada nuevo mensaje que llega es colocado dentro de un *Folder* y le es asignado un número conforme a la secuencia de numeración dentro de éste. El método *getMessageNumber()* sobre un objeto *Message* devuelve su número

asignado. El número asignado al mensaje es válido dentro de una sesión mientras no hay nuevos mensajes o es eliminado alguno; cualquier cambio en el orden de los mensajes modifica la secuencia de numeración de los mismos.

Un cliente puede referenciar a los mensajes almacenados dentro de un *Folder* tanto por el número asignado como por el correspondiente objeto mensaje.

4.3.4 Eventos del objeto *Folder*.

El objeto *Folder* genera eventos para notificar cualquier cambio en él mismo o en su lista de mensajes(cuando llegan mensajes nuevos o es eliminado alguno). El cliente puede registrar métodos para escuchar los eventos generados y ejecutar alguna acción. La clase *Folder* soporta los siguientes eventos:

- *ConnectionEvent*: este evento es generado cuando un *Folder* es abierto o cerrado. Cuando un *Folder* es cerrado(ya sea por que el cliente ha invocado al método *close* o debido a alguna falla en la conexión), todos los componentes de mensajes pertenecientes a este *Folder* son no válidos.

- *Folder Event*: Este evento es generado cuando el cliente crea, elimina o renombra el *Folder*.

- *MessageCountEvent*: este evento notifica que la numeración asociada a los mensajes ha cambiado; las causas de este cambio pueden ser: añadir nuevos mensajes dentro de éste o eliminar mensajes. Para borrar mensajes en un *Folder* es necesario llevar a cabo dos tareas; especificar la bandera de borrado asociada al mensaje con el valor de *TRUE* e invocar al método *expunge*() sobre el objeto *Folder*.

4.4 Protocolos de transporte.

La clase abstracta *Transport* define el envío y el protocolo de transporte para el correo electrónico. Subclases de esta clase implementan *SMTP* y otros protocolos de transporte.

4.4.1 Métodos de la clase *Transport*.

Un objeto *Transport* genera eventos (*Connection Event*) para notificar que se ha establecido la conexión o que ha fallado.

Las implementaciones de *Transport* deben asegurar que el mensaje especificado es de tipo conocido. Si el tipo es conocido, entonces el objeto *Transport* envía el mensaje a los destinatarios especificados. Si el tipo es desconocido entonces, el objeto *Transport* puede intentar reformatear el objeto *Message* para una versión adecuada; si no es posible obtener otra representación es lanzada una excepción indicando dicha falla. Por ejemplo la implementación de transporte *SMTP* reconoce el tipo de datos *MIME(MimeMessages*);

invoca el método *writeTo()* sobre un objeto *MimeMessage* para generar un flujo de *bytes* con formato *RFC822* que es enviado a los destinatarios.

4.4.2 Eventos de la clase *Transport.*

Los clientes pueden registrar métodos para escuchar eventos generados por implementaciones de la clase *Transport*. Dos son los eventos generados; *ConnectionEvent* y *TransportEvent*.

- *ConnectionEvent*: Si la clase transporte obtiene una conexión satisfactoria, entonces genera un evento *ConnectionEvent* con el tipo especificado en OPENED. Si la conexión falla entonces genera el método con el tipo especificado como CLOSED.

- *TransportEvent*: Eventos de transporte. Hay tres tipos de estos eventos: *MESSAGE_DELIVERED*, *MESSAGE_NOT_DELIVERED* Y *MESSAGE_PARTIALLY_DELIVERED*. Los eventos contienen tres arreglos de direcciones: *validSent[],validUnsent[]*, e *invalid[]* que indican las direcciones válidas y no válidas para el mensaje.

- *MESSAGE_DELIVERED*: Cuando el mensaje ha sido enviado satisfactoriamente a todos los recipientes por el protocolo de transporte, *validSent[]* contiene todas las direcciones. *ValidUnsent[]* e *invalid[]* son nulos.

- *MESSAGE_NOT_DELIVERED*: Cuando *validSent[]* es nulo, el mensaje no fue enviado satisfactoriamente a algunos recipientes. *validUnsent[]* puede contener direcciones que son válidas. *invalid[]* puede contener direcciones no válidas.

- *MESSAGE_PARTIALLY_DELIVERED*: Cuando los mensajes fueron enviados de manera satisfactoria a algunos recipientes pero no a todos. *validSent[]* tiene las direcciones de recipientes a los cuales el mensaje fue enviado. *validUnsent[]* tiene las direcciones válidas pero en las cuales el mensaje no fue enviado. *invalid[]* tiene las direcciones no válidas.

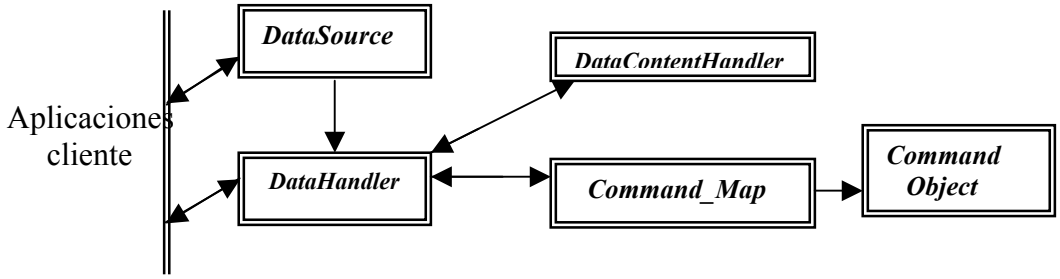
4.5 *Java Beans Activation Framework*

Con la extensión estándar de *Java Beans Activation Framework(JAF),* los desarrolladores que utilizan tecnología *Java* pueden aprovechar servicios estándar para determinar el tipo de alguna pieza de datos, encapsular el acceso a él, descubrir las operaciones disponibles en él, e instanciar el *Bean* adecuado para realizar la operación. Por ejemplo, si se obtiene una imagen del *JPEG*, este marco permite identificar esos de datos como imagen de *JPEG*.

Los *JAF* implementan los siguientes servicios:

1. Determinar el tipo de dato del contenido de mensaje.
2. Encapsular el contenido del mensaje para proveerle acceso .
3. Crear la instancia del componente de software correspondiente a la operación requerida sobre los datos.

Para que los *JAF* implementen dichos servicios conforman un conjunto de clases e interfaces que interactúan entre sí, de esta manera la persona que desarrolla aplicaciones de correo electrónico simplemente los utiliza; no obstante la arquitectura *JAF* permite a los desarrolladores añadir nuevos *Beans* e integrarlos a los existentes.



Esquema 4.2.Mecanismos que integran la arquitectura *JAF*.

La manera en la cual interactúan los componentes de la arquitectura *JAF* para proveer los servicios se describe a continuación de manera breve;

La primera parte consiste en obtener el contenido del mensaje, de ello se encarga la interfaz *DataSource.* Toma el contenido del mensaje y provee métodos que permiten accederlo, asegurando la integridad del mismo, además determina su correspondiente tipo *MIME*; en otras palabras, *encapsula* el objeto que tiene el contenido y devuelve el contenido con su tipo *MIME* correspondiente mediante los métodos definidos en la interfaz *DataSource*.

También se implementan dos clases que utiliza la interfaz *DataSource*:

- *FileDataSource*: permite acceder a los datos cuando éstos se encuentran en un archivo.

- *URLDataSource*: permite acceder a los datos indicando la *URL* donde se encuentran.

La utilidad de la interfaz *DataSource* consiste básicamente en efectuar cuatro actividades:

1. Primera; devolver el contenido del mensaje como un flujo de *bytes* para que pueda ser manipulado.

2. Segunda; devolver un flujo de salida de *bytes* para poder escribir datos en el contenido.

3. Tercera; devolver el tipo *MIME* del contenido del mensaje, (siempre devolverá un tipo válido; si la interfaz no puede determinar el tipo de dato entonces devuelve un tipo que lo indica explícitamente para que la clase que lo reciba lo sepa).

4. Cuarta; devolver el nombre del objeto que contenía el mensaje(en el caso de archivos sólo devuelve le nombre, no regresa la ruta del mismo).

La segunda parte consiste en determinar el comando que será aplicado al contenido del mensaje. Los métodos definidos en la clase *DantaHandler* se utilizan para establecer que comandos pueden ser aplicados, para ello interactúan con la interfaz *CommandMap*.

Hay dos maneras de obtener el contenido del mensaje, en la primera se crea una instancia de *DataHandler* a través de la interfaz *DataSource*(obteniéndose un flujo de bytes para acceder a los datos y el tipo *MIME* de los mismos –lo cual se describió anteriormente-); la segunda manera es crear la instancia *DataHandler* a través de objetos(se especifica un objeto que tiene el contenido del mensaje y su tipo *MIME*).

En el esquema 4.2 se observa que tanto la interfaz *DataSource* como la clase *DataHandler* pueden ser creados(instanciados) por la aplicación cliente para recibir el contenido del mensaje. Otros servicios que provee la clase *DataHandler* es obtener representaciones alternativas del dato (contenido del mensaje) y escribir en el contenido del mensaje.

Cuando el comando ha sido aplicado al objeto que tiene el contenido del mensaje es devuelto a los métodos de la *API JavaMail*.

Algunos métodos de la clase *DataHandler* requieren de la interfaz *DataContentHandler* para llevar a cabo su función:

- El método *getInputStream*(de la clase *DataHandler*)crea un objeto *DataContentHandler* para obtener a partir de un objeto que contiene el dato, un flujo de *bytes* para acceder al mismo, es decir representa el contenido del mensaje como un flujo de *bytes* para que pueda ser manipulado.

- Para representar el dato(contenido del mensaje) de distintas maneras la clase *DataHandler* debe encontrar un contenedor(efectúa una instancia a la interfaz *DataContentHandler*) que corresponda al tipo *MIME* del dato; posteriormente llama al método *getTrasnferDataFlavor* de la interfaz *DataContentHandler* devolviendo los diversos tipos en que puede ser representado el dato(*DataFlavor*). Cuando el dato(contenido del mensaje) ha sido transferido en algún tipo específico el método que lleva a cabo esta tarea devuelve un objeto, utilizando un *DataContenHandler* en este caso para obtener un objeto a partir de un flujo de *bytes*.

- La clase *DataHandler* también provee un método para escribir al contenido de un mensaje, para ello utiliza la interfaz *DataContentHandler*; ya que debe representarse el dato(contenido del mensaje) como flujo de entrada de bytes para ser enviado como argumento del método.

Podemos concluir que la principal utilidad de la interfaz *DataContentHandler* consiste en convertir objetos que contienen el dato(contenido del mensaje) en representaciones de flujos de entrada de *bytes*, y representar flujos de entrada de *byte*s como objetos.

Las interfaces *CommandMap y CommandObject* tienen la siguiente función;

- Una vez que el *DataHandler* tiene el tipo *MIME* que describe el dato(contenido del mensaje) consulta a *CommandMap* las operaciones o comandos disponibles para el tipo de dato, la aplicación solicita comandos disponibles a través del *DantaHandler* y de esta manera especifica un comando de esa lista. *DataHandler* utiliza el *CommandMap* para recuperar el *Bean* asociado con el comando.

Los *Beans* diseñados específicamente para utilizarse con la arquitectura *JAF*, deben implementar la interfaz *CommandObject*. Esta interfaz provee acceso directo a los métodos del *DataHandler*. Una vez efectuada la instanciación, el *Bean* toma una cadena que indica el comando seleccionado y el objeto *DataHandlder* administra el contenido. A través de un objeto *DataHandler* la *API JavaMail* recupera el dato.

6. Análisis y Diseño

El proceso de reunión de requisitos se intensifica y se centra especialmente en el software. Para comprender la naturaleza de el (los) programa(s) a construirse, el analista del software debe comprender el dominio de información del software, así como la función requerida, comportamiento, rendimiento, e interconexión.

El diseño del software es realmente un proceso de muchos pasos que se centra en cuatro atributos distintos de un programa: estructura de datos, arquitectura del software, representaciones de interfaz y detalle procedimental(algoritmo). El proceso de diseño traduce requisitos en una representación del software que se pueda evaluar por calidad antes de que comience la generación de código.

6.1.1 Análisis de requerimientos del sistema

El sistema a desarrollar permitirá al usuario aplicar filtros sobre los mensajes contenidos en el buzón de correo electrónico, cuyo protocolo de almacenamiento sea *POP3 o IMAP*, de esta manera el usuario tendrá una herramienta que le facilitará la administración del correo electrónico.

El sistema define cuatro tareas:

- *Diagnóstico*: Detectar cuales mensajes de correo podrían tener falsos de remitente.
- *Aplicar Fitltros*: Eliminar mensajes especificando criterios con base en algunos de sus atributos; tamaño, asunto y remitente.
- *Envío Automático:* Responder automáticamente a peticiones de archivos, con base en la línea asunto.
- *Envío:* Enviar mensajes.

Diagnóstico

Cuando el usuario solicite esta opción, el sistema obtendrá la línea *Received* de cada mensaje para comparar el dominio de la dirección de correo electrónico del remitente, con el dominio del servidor de correo saliente del remitente, si ambos dominios son iguales, entonces el correo se considerará legítimo; en caso contrario será considerado no legítimo.

El sistema mostrará a través de una interfaz gráfica el listado de los mensajes cuyos datos de remitente podrían ser falsos, de esta manera el usuario podrá elegir cualquiera de la lista para leer los datos de la línea *Received*, y decidir si lo elimina. También estará disponible la opción eliminar todos los mensajes.

Aplicar Filtros

Para este caso la aplicación desplegará un listado con todos los mensajes contenidos en el buzón; el usuario podrá especificar criterios para de filtrado con base en algunos de los atributos de los mensajes(tamaño, dominio, remitente y asunto),a demás de estar disponible la opción de eliminar todos los mensajes. Por ejemplo, obtener todos los mensajes cuyo tamaño sea mayor o igual que 300 *bytes*.

Envío Automático

El envío automatizado de mensajes permite al usuario enviar mensajes con archivos adjuntos, la aplicación revisará cada línea asunto de todos los mensajes, cada petición encontrada será respondida (adjuntando los archivos solicitados), el usuario únicamente deberá indicar la ruta absoluta del directorio donde estén almacenados los archivos que podrían ser solicitados.

Tanto la opción de envío automático como la opción envío tendrán un límite de 3*MB* para archivos adjuntos.

La sintaxis para solicitar archivos es:

Archivo*=arch1.extensión+arch2.extensión+.....+archn.extensión*

Las restricciones son;

Pueden utilizarse letras minúsculas o mayúsculas.
Los nombres de los archivos deberán indicarse con nombre y extensión

Envío.

Esta opción permite enviar mensajes de correo electrónico, incluyendo archivos adjuntos. Únicamente podrá enviarse como cuerpo del mensaje texto simple.

6.1.2 Diagrama de *casos de uso*

Un caso de uso es una interacción típica entre un usuario y un sistema de cómputo. El caso de uso capta alguna función visible para el usuario; en su forma más simple, el caso de uso se obtiene hablando con los usuarios habituales y analizando con ellos las distintas cosas que deseen hacer con el sistema. Ver figura 6.1

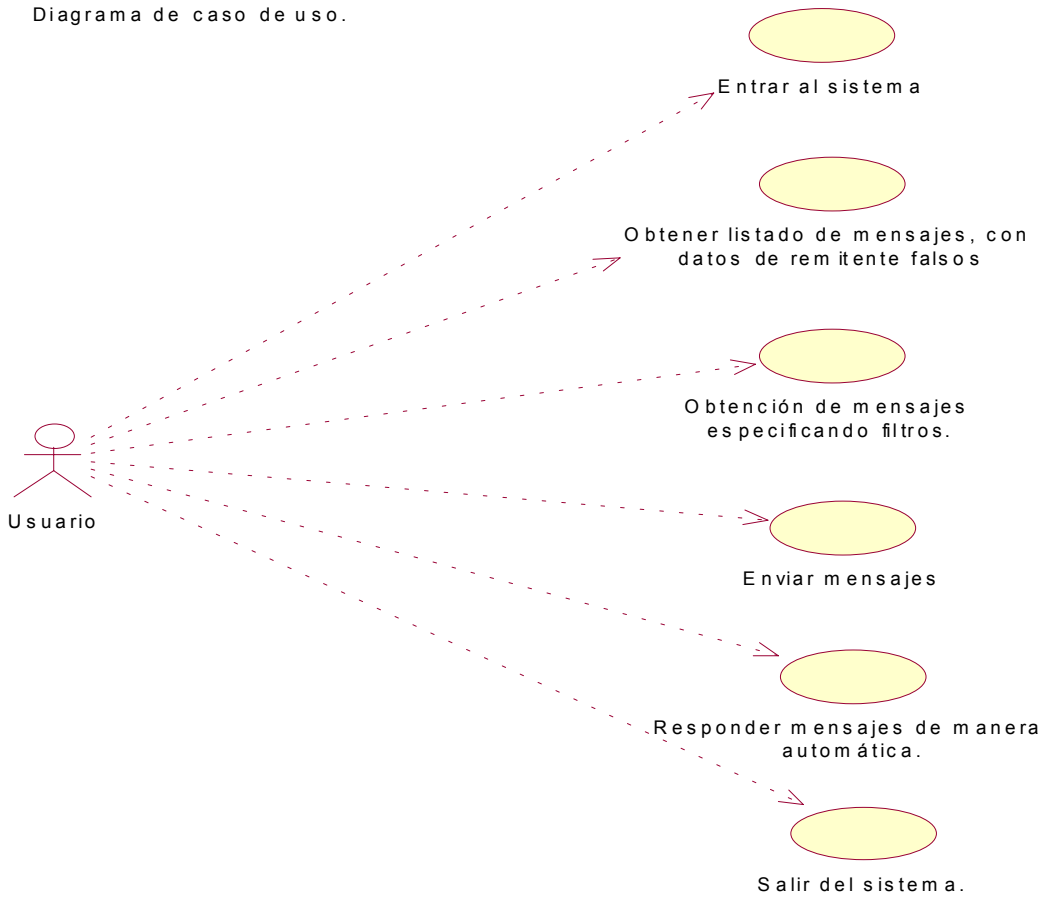


Figura 6.1 Diagrama de casos de uso

6.2.1 Diagrama de secuencia

Los diagramas de secuencia son modelos que describen la manera en que colaboran grupos de objetos para cierto comportamiento.

Un diagrama de secuencia capta el comportamiento de un solo caso de uso. El diagrama muestra cierto número de ejemplos de objetos y los mensajes que se pasan entre estos objetos dentro del caso de uso.

La línea vertical se llama línea de vida del objeto, la cual representa la vida del objeto durante la interacción.

Cada mensaje se representa mediante una flecha entre las líneas de vida de dos objetos. El orden en el que se dan estos objetos transcurre de arriba hacia abajo. Cada mensaje es etiquetado por lo menos con el nombre del mensaje; pueden incluirse también los

argumentos y alguna información de control, y se puede mostrar la autodelegación, que es un mensaje de vuelta a la misma línea de vida.

El diagrama de secuencia de la figura 6.2 muestra de manera general el procedimiento de interacción entre los objetos para llevar a cabo las tareas descritas tanto en el diagrama de casos de uso como en el análisis de requerimientos, no describe detalladamente los procedimientos requeridos para efectuar cada tarea.

Primeramente es solicitada la conexión al servidor de correo entrante en el cual el usuario tiene asignado un buzón (espacio en el sistema de almacenamiento del servidor) para su correo electrónico; si la conexión se establece, entonces el sistema muestra un menú con cuatro opciones, correspondientes a las tareas de *diagnostico(), envío automático()*, aplicar *filtros()* y *envío de mensajes()*.

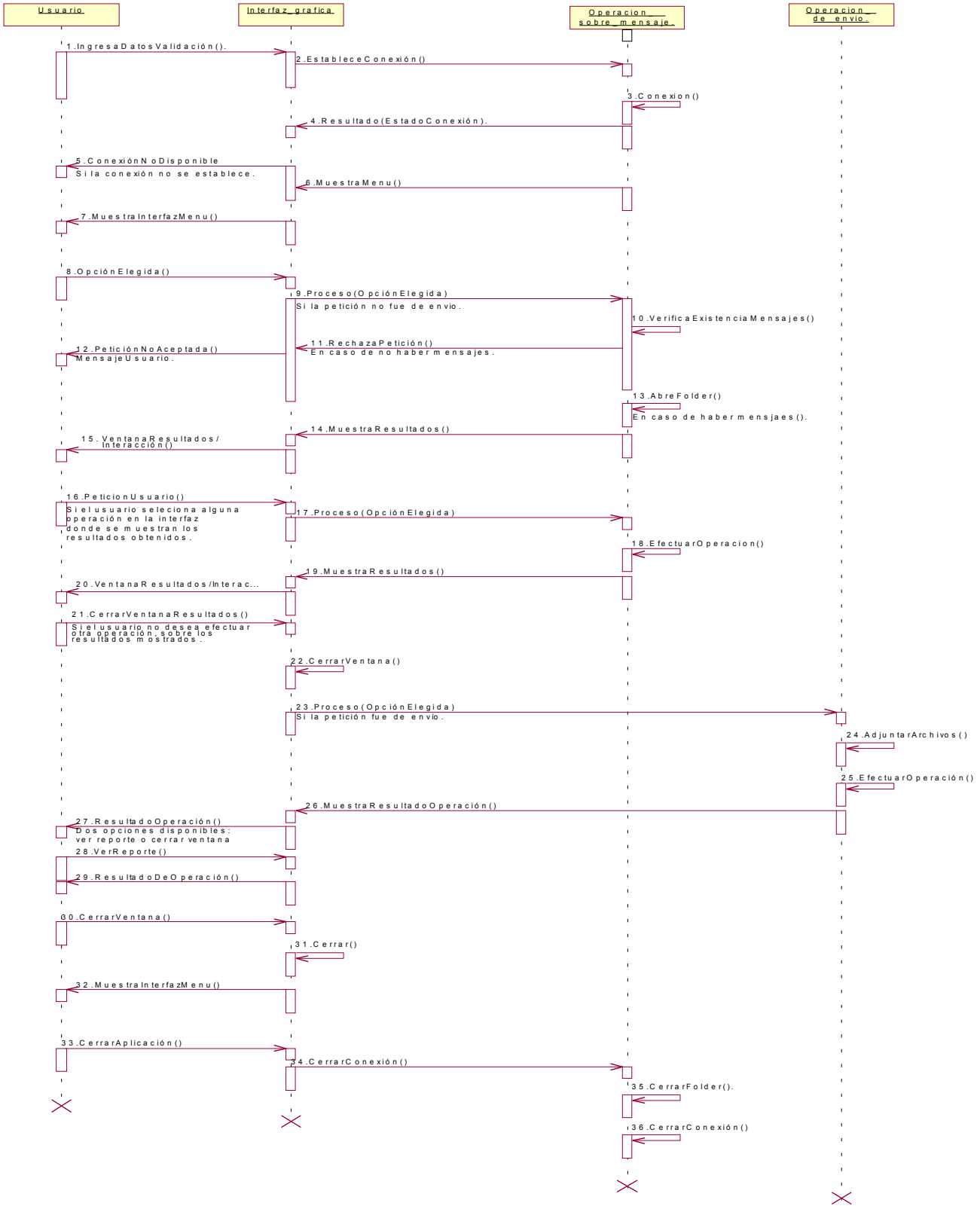
El diagrama indica que para las opciones *diagnóstico(), envío automático(),* y aplicar filtros() se requiere la existencia de mensajes, de no cumplirse esta condición será devuelto un mensaje al usuario indicando el motivo por el cual no es posible llevar a cabo la opción seleccionada. Si hay mensajes en el buzón, se efectuarán los procesos *AbreFolder()* y *EfectuarOperación()*.

El método *AbreFolder()* lleva a cabo la apertura de la -carpeta bandeja de entrada- (*INBOX*), en la cual son almacenados todos los mensajes de correo electrónico que han sido enviados al propietario de la cuenta de correo electrónico(algunas interfaces para administrar el correo electrónico ofrecen la opción de crear carpetas, éstas únicamente representan otra forma de acceso a los mensajes de correo). Cuando el fólder ha sido abierto, el método *EfectuarOperacion()* lleva a cabo las operaciones indicadas sobre los mensajes(obtener atributos y eliminar mensajes principalmente), las cuales serán especificadas de acuerdo a la opción elegida del menú.

El diagrama también indica que los resultados obtenidos en cada operación se muestran mediante una interfaz gráfica, la cual permite al usuario elegir alguna operación sobre estos. En el caso de la opción *Diagnostico(),* el usuario podrá elegir dos opciones: eliminar el mensaje, o leer el reporte generado por la aplicación acerca del mensaje(datos del remitente y línea *Received*).

En caso de seleccionar la opción *envío automático()* el sistema mostrará en pantalla la lista de los mensajes enviados; en esta opción serán eliminados automáticamente los mensajes después de que han sido contestados.

Para el envío de mensajes el diagrama indica simplemente la llamada al objeto envío, el cual efectúa el procedimiento para adjuntar archivos y enviar el mensaje; en caso de ocurrir algún problema durante el envío el sistema lo indicará mediante un cuadro de diálogo.


Usuario
Interfaz_grafica
Operacion_sobre_mensaje.
Operacion_de_envio.
1.IngresaDatosValidación().
2.EstableceConexión()
3.Conexion()
4.Resultado(EstadoConexión).
5.ConexiónNoDisponible
Si la conexión no se establece.
6.MuestraMenu()
7.MuestraInterfazMenu()
8.OpciónElegida()
9.Proceso(OpciónElegida)
Si la petición no fue de envio.
10.VerificaExistenciaMensajes()
11.RechazaPetición()
En caso de no haber mensajes.
12.PeticiónNoAceptada()
MensajeUsuario.
13.AbreFolder()
En caso de haber mensjaes().
14.MuestraResultados()
15.VentanaResultados/
Interacción()
16.PeticionUsuario()
Si el usuario seleciona alguna
operación en la interfaz
donde se muestran los
resultados obtenidos.
17.Proceso(OpciónElegida)
18.EfectuarOperacion()
19.MuestraResultados()
20.VentanaResultados/Interac...
21.CerrarVentanaResultados()
Si el usuario no desea efectuar
otra operación sobre los
resultados mostrados.
22.CerrarVentana()
23.Proceso(OpciónElegida)
Si la petición fue de envio.
24.AdjuntarArchivos()
25.EfectuarOperación()
26.MuestraResultadoOperación()
27.ResultadoOperación()
Dos opciones disponibles:
ver reporte o cerrar ventana
28.VerReporte()
29.ResultadoDeOperación()
30.CerrarVentana()
31.Cerrar()
32.MuestraInterfazMenu()
33.CerrarAplicación()
34.CerrarConexión()
35.CerrarFolder().
36.CerrarConexión()

2.2.2 Diagrama de clases

En el diagrama de clases(figura 6.3) el objeto "ingreso_buzón" fue considerado dentro de la clase venta_interacción, de esta manera tenemos cuatro clases, tres de éstas se relacionan entre sí (ventana_interaccion, operación_sobre_mensaje, y envío). Las relaciones se leen de la siguiente manera; cada *operación sobre los mensajes es* solicitada a través de una *ventana_de_interaccion,* y una *ventana_de_interacción* es utilizada para seleccionar una o mas *operaciones sobre los mensajes.*

El diagrama de clases está organizado en cuatro clases, cada clase contiene los métodos relacionados con base en su objetivo, por ejemplo en la clase *operación_sobre_mensaje* están contenidos todos los métodos que operan directamente en el buzón de correo del usuario(conexión(), eliminar mensajes(), ...), la clase *ventana_interacción* contiene los métodos para desplegar en pantalla los datos solicitados por el usuario al seleccionar alguna opción del menú, los métodos de la clase *operación_de_envío* no forman parte de la clase *operación_sobre_mensaje,* ya que no necesitan obtener datos que requieran alguna operación en el buzón de correo, sí hay relación entre ambas clases(*operación_de_envío y operación_sobre_mensaje*) por que la opción *EnvioAutomatizado()* debe buscar en el buzón de correo los mensajes cuya línea asunto especifique la petición de archivos, posteriormente efectuará el procedimiento para envío(incluyendo la operación de adjuntar archivos), para ello esta clase utiliza los métodos definidos en *operación_envío.*

2.2.3 Diagrama de estados

En la figura 6.31 se muestra el diagrama de estados. Se determinó que el buzón de correo cambia de estado respecto del tiempo, ya que las operaciones efectuadas sobre este así lo indican. El buzón está en el estado *Folder_Abierto* cuando ha sido ejecutada la instrucción *AbreFolder()*, posteriormente efectúa la operación *CuentaMensjaes()* para definir si estará en el estado *Espera_Petición* o *Folder_Vacío*.

Si el estado es *Espera_Petición*, el diagrama indica que recibirá una petición de un evento externo, lo cual representa simplemente alguna petición que el usuario haga mediante la interfaz gráfica. Si la petición fue *BorrarMensajes()* el estado del fólder será *Procesa_Petición*, posteriormente la instrucción *CerrarFolder()* lo llevará a *Folder_Cerrado()*, en dicho estado se eliminarán los mensajes que el usuario haya indicado.

Si la petición fue *Obtiene_Atributos()* únicamente llegará al estado *Procesa_Petición*, posteriormente regresará al estado *Espera_Petición()*.

Cada vez que el fólder llega al estado de *Folder_Abierto* se ejecuta la instrucción *Cuenta_Mensajes()*, si aún hay mensajes el fólder llegará al estado *Espera_Petición*, en caso contrario llegará al de *Folder_Vacío*.

6.2.3 Diseño

6.2.4 Diagrama de componentes

En esta etapa se lleva a cabo el diagrama de componentes, el cual describe tanto componentes de *software* como las dependencias con otros componentes, con el fin de representar la estructura del código fuente. El termino componentes de *software* se refiere a componentes del código fuente, componentes binarios y los componentes ejecutables.

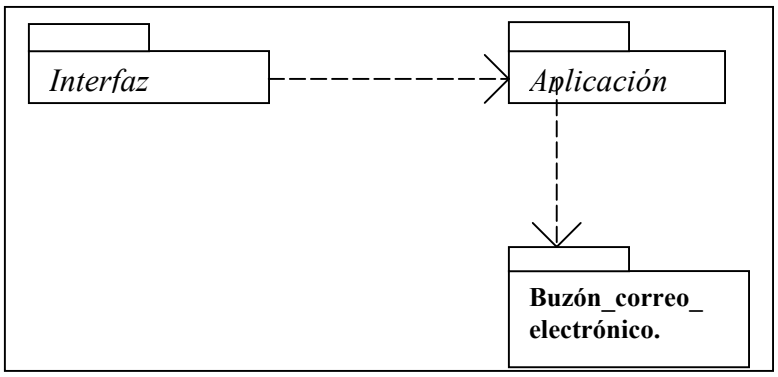


Figura 6-4. Diagrama de componentes

En el esquema de la figura 6.4 se muestran tres carpetas , los cuales representan paquetes de clases. Cada paquete se liga con otros mediante dependencias, representadas por flechas con líneas discontinuas, cuyo origen es del componente dependiente.

También podemos observar en el diagrama 6.4 de manera general la dependencia entre los componentes que forman parte de la aplicación. Anteriormente en el diagrama de secuencia se describió que el usuario podrá seleccionar a través de una interfaz gráfica alguna de las opciones, posteriormente cada resultado obtenido será mostrado mediante una interfaz gráfica, el diagrama de componentes afirma esta descripción, ya que la interfaz gráfica depende de la aplicación, y la aplicación depende del buzón de correo electrónico, por lo tanto cada petición del usuario será procesada por algún métodos de alguna clase del paquete aplicación; cada clase del paquete aplicación será determinada por las especificaciones del buzón de correo electrónico.

Cada paquete se asocia con un diagrama de componentes para representar las clases contenidas internamente. En el diagrama de la figura 6.5 se muestran los componentes de software para el paquete Interfaz gráfica, el cual presenta tres componentes de software *Ventana de interacción, Menú e Ingreso*. A partir del diagrama podemos establecer que el componente *Menú,* determinará que interfaz gráfica se presentará al usuario, ya que *Ventana de Interacción* depende de la opción elegida en el componente *Menú.* También se observa el componente *Ingreso,* el cual es una interfaz gráfica, la cual se mostrará al inicio del sistema para obtener los datos del usuario.

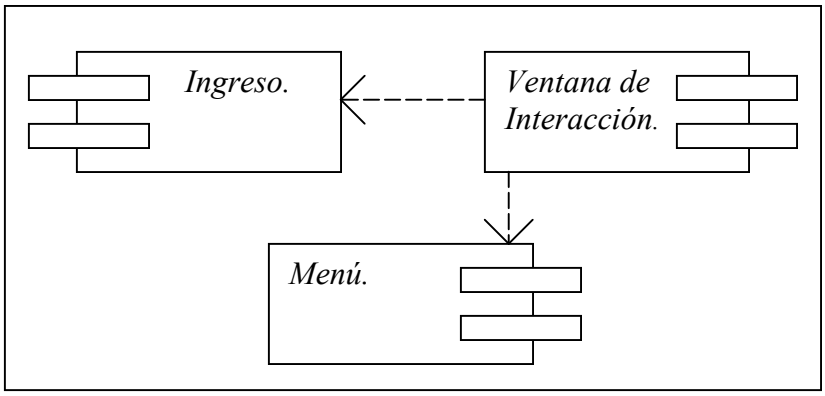


Figura 6.5
Diagrama de componentes para el paquete Interfaz gráfica

En el caso del paquete *Aplicación* simplemente agrupa todas las clases del código fuente del sistema.

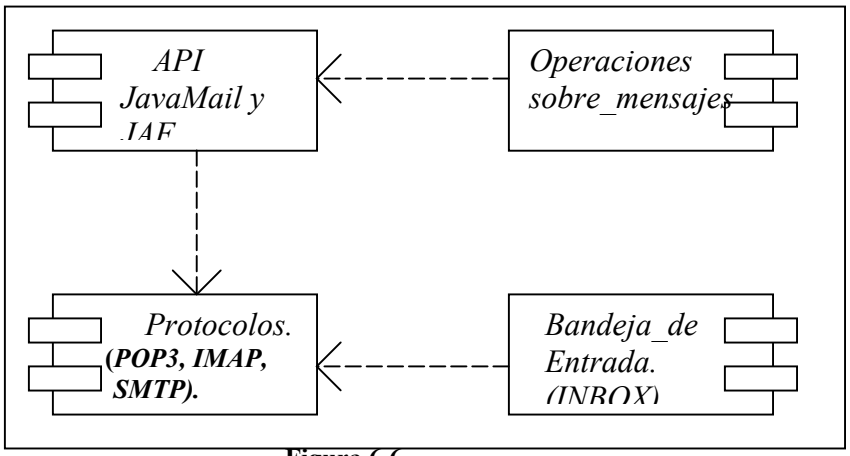


Figura 6.6
Diagrama de componentes para el paquete Buzón de correo electrónico

El componente buzón de correo electrónico básicamente es un directorio, en el medio de almacenamiento de un servidor de correo entrante que recibe mensajes de correo electrónico para determinado dominio, donde cada directorio corresponde a una cuenta de correo electrónico para algún usuario.

Para enviar mensajes de correo electrónico se utilizan protocolos de transporte, los cuales definen procedimientos especificados para ello. El usuario no necesita conocer el funcionamiento de dichos protocolos para enviar mensajes de correo electrónico.

Para obtener los mensajes de correo electrónico el usuario utiliza en forma transparente protocolos de almacenamiento(*POP3 e IMAP*), los cuales proporcionan opciones de lectura y borrado de los mensajes principalmente.

Cada interfaz desarrollada para administrar el correo electrónico, implementa procedimientos para interactuar con los protocolos tanto de transporte como de almacenamiento del sistema de correo electrónico para el que fue diseñada, con el fin de aplicar operaciones sobre los mensajes (leer, borrar, obtener atributos, enviar, adjuntar archivos, etcétera).

En esta aplicación se utilizó la *API JavaMail*. En el diagrama 6.6 podemos observar que el componente operaciones sobre mensajes depende de los componentes que provee la *API JavaMail*; la cual depende de los procedimientos definidos en el componente protocolos, ya que las clases definidas en *JavaMail,* están diseñadas de acuerdo con las operaciones definidas para sistemas de correo electrónico basados en estándares abiertos, dicha *API* no define alguna clase cuyos métodos, efectúen alguna tarea que no esté definida en la especificación de los protocolos. Finalmente se observa que el componente bandeja de entrada depende del componente protocolos, ya que este último define algunas operaciones aplicables a los mensajes de correo electrónico en este diseño(otras opciones como filtrar no están definida en el componente protocolos).

6. Análisis y Diseño

El proceso de reunión de requisitos se intensifica y se centra especialmente en el software. Para comprender la naturaleza de el (los) programa(s) a construirse, el analista del software debe comprender el dominio de información del software, así como la función requerida, comportamiento, rendimiento, e interconexión.

El diseño del software es realmente un proceso de muchos pasos que se centra en cuatro atributos distintos de un programa: estructura de datos, arquitectura del software, representaciones de interfaz y detalle procedimental(algoritmo). El proceso de diseño traduce requisitos en una representación del software que se pueda evaluar por calidad antes de que comience la generación de código.

6.1.1 Análisis de requerimientos del sistema

El sistema a desarrollar permitirá al usuario aplicar filtros sobre los mensajes contenidos en el buzón de correo electrónico, cuyo protocolo de almacenamiento sea *POP3 o IMAP*, de esta manera el usuario tendrá una herramienta que le facilitará la administración del correo electrónico.

El sistema define cuatro tareas:

- *Diagnóstico*: Detectar cuales mensajes de correo podrían tener falsos de remitente.
- *Aplicar Fitltros*: Eliminar mensajes especificando criterios con base en algunos de sus atributos; tamaño, asunto y remitente.
- *Envío Automático:* Responder automáticamente a peticiones de archivos, con base en la línea asunto.
- *Envío:* Enviar mensajes.

Diagnóstico

Cuando el usuario solicite esta opción, el sistema obtendrá la línea *Received* de cada mensaje para comparar el dominio de la dirección de correo electrónico del remitente, con el dominio del servidor de correo saliente del remitente, si ambos dominios son iguales, entonces el correo se considerará legítimo; en caso contrario será considerado no legítimo.

El sistema mostrará a través de una interfaz gráfica el listado de los mensajes cuyos datos de remitente podrían ser falsos, de esta manera el usuario podrá elegir cualquiera de la lista para leer los datos de la línea *Received*, y decidir si lo elimina. También estará disponible la opción eliminar todos los mensajes.

Aplicar Filtros

Para este caso la aplicación desplegará un listado con todos los mensajes contenidos en el buzón; el usuario podrá especificar criterios para de filtrado con base en algunos de los atributos de los mensajes(tamaño, dominio, remitente y asunto),a demás de estar disponible la opción de eliminar todos los mensajes. Por ejemplo, obtener todos los mensajes cuyo tamaño sea mayor o igual que 300 *bytes*.

Envío Automático

El envío automatizado de mensajes permite al usuario enviar mensajes con archivos adjuntos, la aplicación revisará cada línea asunto de todos los mensajes, cada petición encontrada será respondida (adjuntando los archivos solicitados), el usuario únicamente deberá indicar la ruta absoluta del directorio donde estén almacenados los archivos que podrían ser solicitados.

Tanto la opción de envío automático como la opción envío tendrán un límite de 3*MB* para archivos adjuntos.

La sintaxis para solicitar archivos es:

Archivo*=arch1.extensión+arch2.extensión+.....+archn.extensión*

Las restricciones son;

Pueden utilizarse letras minúsculas o mayúsculas.
Los nombres de los archivos deberán indicarse con nombre y extensión

Envío.

Esta opción permite enviar mensajes de correo electrónico, incluyendo archivos adjuntos. Únicamente podrá enviarse como cuerpo del mensaje texto simple.

6.1.2 Diagrama de *casos de uso*

Un caso de uso es una interacción típica entre un usuario y un sistema de cómputo. El caso de uso capta alguna función visible para el usuario; en su forma más simple, el caso de uso se obtiene hablando con los usuarios habituales y analizando con ellos las distintas cosas que deseen hacer con el sistema. Ver figura 6.1

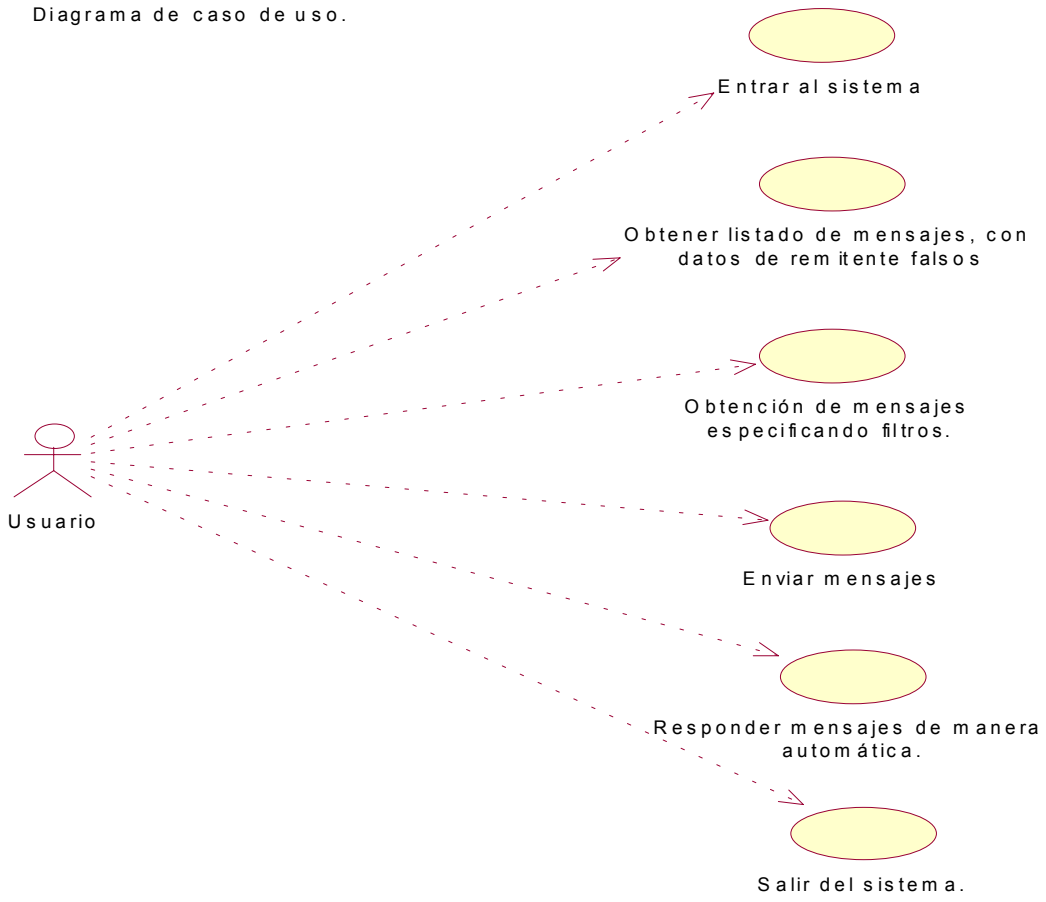


Figura 6.1 Diagrama de casos de uso

6.2.1 Diagrama de secuencia

Los diagramas de secuencia son modelos que describen la manera en que colaboran grupos de objetos para cierto comportamiento.

Un diagrama de secuencia capta el comportamiento de un solo caso de uso. El diagrama muestra cierto número de ejemplos de objetos y los mensajes que se pasan entre estos objetos dentro del caso de uso.

La línea vertical se llama línea de vida del objeto, la cual representa la vida del objeto durante la interacción.

Cada mensaje se representa mediante una flecha entre las líneas de vida de dos objetos. El orden en el que se dan estos objetos transcurre de arriba hacia abajo. Cada mensaje es etiquetado por lo menos con el nombre del mensaje; pueden incluirse también los

argumentos y alguna información de control, y se puede mostrar la autodelegación, que es un mensaje de vuelta a la misma línea de vida.

El diagrama de secuencia de la figura 6.2 muestra de manera general el procedimiento de interacción entre los objetos para llevar a cabo las tareas descritas tanto en el diagrama de casos de uso como en el análisis de requerimientos, no describe detalladamente los procedimientos requeridos para efectuar cada tarea.

Primeramente es solicitada la conexión al servidor de correo entrante en el cual el usuario tiene asignado un buzón (espacio en el sistema de almacenamiento del servidor) para su correo electrónico; si la conexión se establece, entonces el sistema muestra un menú con cuatro opciones, correspondientes a las tareas de *diagnostico(), envío automático()*, aplicar *filtros()* y *envío de mensajes()*.

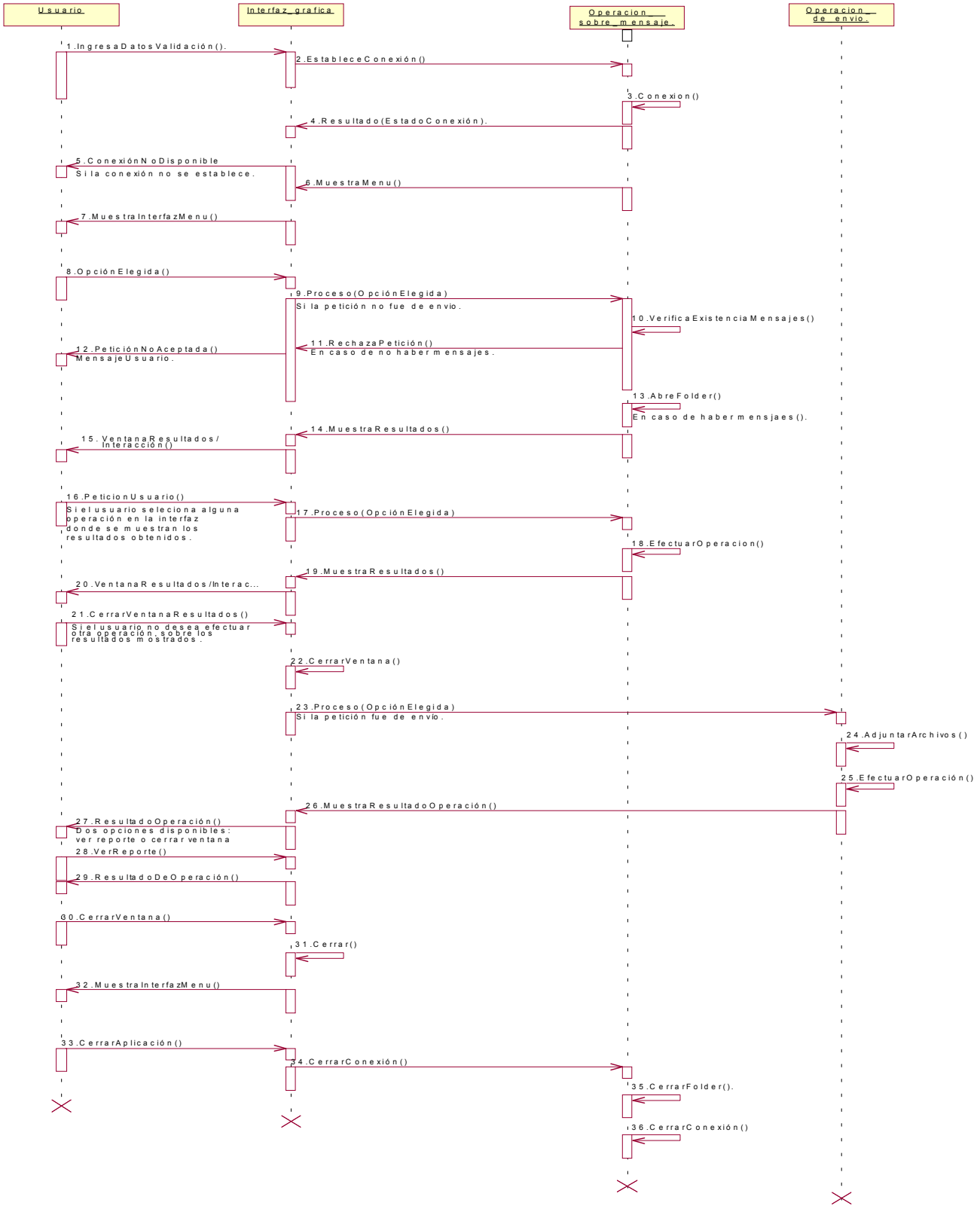
El diagrama indica que para las opciones *diagnóstico(), envío automático(),* y aplicar filtros() se requiere la existencia de mensajes, de no cumplirse esta condición será devuelto un mensaje al usuario indicando el motivo por el cual no es posible llevar a cabo la opción seleccionada. Si hay mensajes en el buzón, se efectuarán los procesos *AbreFolder()* y *EfectuarOperación()*.

El método *AbreFolder()* lleva a cabo la apertura de la -carpeta bandeja de entrada- (*INBOX*), en la cual son almacenados todos los mensajes de correo electrónico que han sido enviados al propietario de la cuenta de correo electrónico(algunas interfaces para administrar el correo electrónico ofrecen la opción de crear carpetas, éstas únicamente representan otra forma de acceso a los mensajes de correo). Cuando el fólder ha sido abierto, el método *EfectuarOperacion()* lleva a cabo las operaciones indicadas sobre los mensajes(obtener atributos y eliminar mensajes principalmente), las cuales serán especificadas de acuerdo a la opción elegida del menú.

El diagrama también indica que los resultados obtenidos en cada operación se muestran mediante una interfaz gráfica, la cual permite al usuario elegir alguna operación sobre estos. En el caso de la opción *Diagnostico(),* el usuario podrá elegir dos opciones: eliminar el mensaje, o leer el reporte generado por la aplicación acerca del mensaje(datos del remitente y línea *Received*).

En caso de seleccionar la opción *envío automático()* el sistema mostrará en pantalla la lista de los mensajes enviados; en esta opción serán eliminados automáticamente los mensajes después de que han sido contestados.

Para el envío de mensajes el diagrama indica simplemente la llamada al objeto envío, el cual efectúa el procedimiento para adjuntar archivos y enviar el mensaje; en caso de ocurrir algún problema durante el envío el sistema lo indicará mediante un cuadro de diálogo.


Usuario
Interfaz_grafica
Operacion_sobre_mensaje.
Operacion_de_envio.
1.IngresaDatosValidación().
2.EstableceConexión()
3.Conexion()
4.Resultado(EstadoConexión).
5.ConexiónNoDisponible
Si la conexión no se establece.
6.MuestraMenu()
7.MuestraInterfazMenu()
8.OpciónElegida()
9.Proceso(OpciónElegida)
Si la petición no fue de envio.
10.VerificaExistenciaMensajes()
11.RechazaPetición()
En caso de no haber mensajes.
12.PeticiónNoAceptada()
MensajeUsuario.
13.AbreFolder()
En caso de haber mensjaes().
14.MuestraResultados()
15.VentanaResultados/
Interacción()
16.PeticionUsuario()
Si el usuario seleciona alguna
operación en la interfaz
donde se muestran los
resultados obtenidos.
17.Proceso(OpciónElegida)
18.EfectuarOperacion()
19.MuestraResultados()
20.VentanaResultados/Interac...
21.CerrarVentanaResultados()
Si el usuario no desea efectuar
otra operación sobre los
resultados mostrados.
22.CerrarVentana()
23.Proceso(OpciónElegida)
Si la petición fue de envio.
24.AdjuntarArchivos()
25.EfectuarOperación()
26.MuestraResultadoOperación()
27.ResultadoOperación()
Dos opciones disponibles:
ver reporte o cerrar ventana
28.VerReporte()
29.ResultadoDeOperación()
30.CerrarVentana()
31.Cerrar()
32.MuestraInterfazMenu()
33.CerrarAplicación()
34.CerrarConexión()
35.CerrarFolder().
36.CerrarConexión()

2.2.2 Diagrama de clases

En el diagrama de clases(figura 6.3) el objeto "ingreso_buzón" fue considerado dentro de la clase venta_interacción, de esta manera tenemos cuatro clases, tres de éstas se relacionan entre sí (ventana_interaccion, operación_sobre_mensaje, y envío). Las relaciones se leen de la siguiente manera; cada *operación sobre los mensajes es* solicitada a través de una *ventana_de_interaccion,* y una *ventana_de_interacción* es utilizada para seleccionar una o mas *operaciones sobre los mensajes.*

El diagrama de clases está organizado en cuatro clases, cada clase contiene los métodos relacionados con base en su objetivo, por ejemplo en la clase *operación_sobre_mensaje* están contenidos todos los métodos que operan directamente en el buzón de correo del usuario(conexión(), eliminar mensajes(), ...), la clase *ventana_interacción* contiene los métodos para desplegar en pantalla los datos solicitados por el usuario al seleccionar alguna opción del menú, los métodos de la clase *operación_de_envío* no forman parte de la clase *operación_sobre_mensaje,* ya que no necesitan obtener datos que requieran alguna operación en el buzón de correo, sí hay relación entre ambas clases(*operación_de_envío y operación_sobre_mensaje*) por que la opción *EnvioAutomatizado()* debe buscar en el buzón de correo los mensajes cuya línea asunto especifique la petición de archivos, posteriormente efectuará el procedimiento para envío(incluyendo la operación de adjuntar archivos), para ello esta clase utiliza los métodos definidos en *operación_envío.*

2.2.3 Diagrama de estados

En la figura 6.31 se muestra el diagrama de estados. Se determinó que el buzón de correo cambia de estado respecto del tiempo, ya que las operaciones efectuadas sobre este así lo indican. El buzón está en el estado *Folder_Abierto* cuando ha sido ejecutada la instrucción *AbreFolder()*, posteriormente efectúa la operación *CuentaMensjaes()* para definir si estará en el estado *Espera_Petición* o *Folder_Vacío*.

Si el estado es *Espera_Petición*, el diagrama indica que recibirá una petición de un evento externo, lo cual representa simplemente alguna petición que el usuario haga mediante la interfaz gráfica. Si la petición fue *BorrarMensajes()* el estado del fólder será *Procesa_Petición*, posteriormente la instrucción *CerrarFolder()* lo llevará a *Folder_Cerrado()*, en dicho estado se eliminarán los mensajes que el usuario haya indicado.

Si la petición fue *Obtiene_Atributos()* únicamente llegará al estado *Procesa_Petición*, posteriormente regresará al estado *Espera_Petición()*.

Cada vez que el fólder llega al estado de *Folder_Abierto* se ejecuta la instrucción *Cuenta_Mensajes()*, si aún hay mensajes el fólder llegará al estado *Espera_Petición*, en caso contrario llegará al de *Folder_Vacío*.

6.2.3 Diseño

6.2.4 Diagrama de componentes

En esta etapa se lleva a cabo el diagrama de componentes, el cual describe tanto componentes de *software* como las dependencias con otros componentes, con el fin de representar la estructura del código fuente. El termino componentes de *software* se refiere a componentes del código fuente, componentes binarios y los componentes ejecutables.

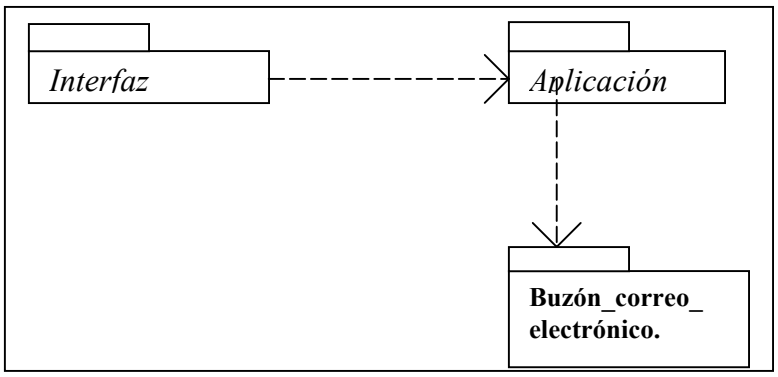


Figura 6-4. Diagrama de componentes

En el esquema de la figura 6.4 se muestran tres carpetas , los cuales representan paquetes de clases. Cada paquete se liga con otros mediante dependencias, representadas por flechas con líneas discontinuas, cuyo origen es del componente dependiente.

También podemos observar en el diagrama 6.4 de manera general la dependencia entre los componentes que forman parte de la aplicación. Anteriormente en el diagrama de secuencia se describió que el usuario podrá seleccionar a través de una interfaz gráfica alguna de las opciones, posteriormente cada resultado obtenido será mostrado mediante una interfaz gráfica, el diagrama de componentes afirma esta descripción, ya que la interfaz gráfica depende de la aplicación, y la aplicación depende del buzón de correo electrónico, por lo tanto cada petición del usuario será procesada por algún métodos de alguna clase del paquete aplicación; cada clase del paquete aplicación será determinada por las especificaciones del buzón de correo electrónico.

Cada paquete se asocia con un diagrama de componentes para representar las clases contenidas internamente. En el diagrama de la figura 6.5 se muestran los componentes de software para el paquete Interfaz gráfica, el cual presenta tres componentes de software *Ventana de interacción, Menú e Ingreso*. A partir del diagrama podemos establecer que el componente *Menú,* determinará que interfaz gráfica se presentará al usuario, ya que *Ventana de Interacción* depende de la opción elegida en el componente *Menú.* También se observa el componente *Ingreso,* el cual es una interfaz gráfica, la cual se mostrará al inicio del sistema para obtener los datos del usuario.

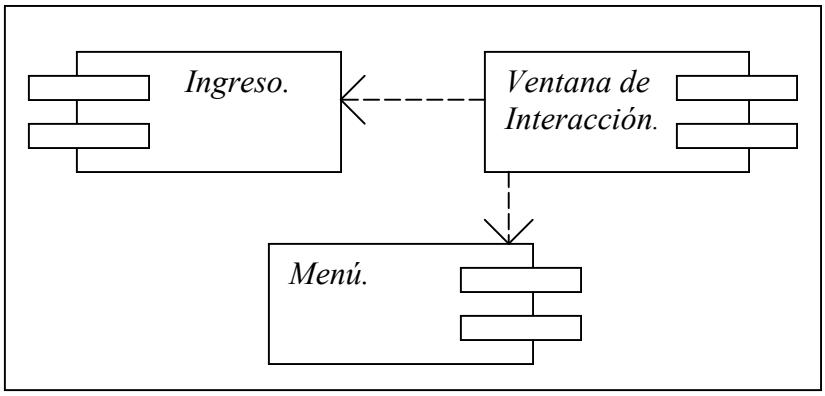


Figura 6.5
Diagrama de componentes para el paquete Interfaz gráfica

En el caso del paquete *Aplicación* simplemente agrupa todas las clases del código fuente del sistema.

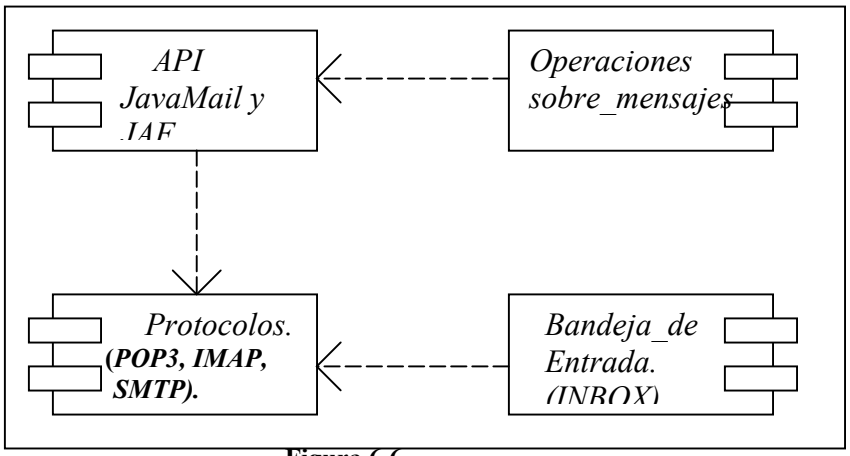


Figura 6.6
Diagrama de componentes para el paquete Buzón de correo electrónico

El componente buzón de correo electrónico básicamente es un directorio, en el medio de almacenamiento de un servidor de correo entrante que recibe mensajes de correo electrónico para determinado dominio, donde cada directorio corresponde a una cuenta de correo electrónico para algún usuario.

Para enviar mensajes de correo electrónico se utilizan protocolos de transporte, los cuales definen procedimientos especificados para ello. El usuario no necesita conocer el funcionamiento de dichos protocolos para enviar mensajes de correo electrónico.

Para obtener los mensajes de correo electrónico el usuario utiliza en forma transparente protocolos de almacenamiento(*POP3 e IMAP*), los cuales proporcionan opciones de lectura y borrado de los mensajes principalmente.

Cada interfaz desarrollada para administrar el correo electrónico, implementa procedimientos para interactuar con los protocolos tanto de transporte como de almacenamiento del sistema de correo electrónico para el que fue diseñada, con el fin de aplicar operaciones sobre los mensajes (leer, borrar, obtener atributos, enviar, adjuntar archivos, etcétera).

En esta aplicación se utilizó la *API JavaMail*. En el diagrama 6.6 podemos observar que el componente operaciones sobre mensajes depende de los componentes que provee la *API JavaMail*; la cual depende de los procedimientos definidos en el componente protocolos, ya que las clases definidas en *JavaMail,* están diseñadas de acuerdo con las operaciones definidas para sistemas de correo electrónico basados en estándares abiertos, dicha *API* no define alguna clase cuyos métodos, efectúen alguna tarea que no esté definida en la especificación de los protocolos. Finalmente se observa que el componente bandeja de entrada depende del componente protocolos, ya que este último define algunas operaciones aplicables a los mensajes de correo electrónico en este diseño(otras opciones como filtrar no están definida en el componente protocolos).

7. Programación del sistema

Debido a que el código fuente es demasiado extenso, se muestra sólo un fragmento con lo más importante del sistema.

Listado de paquetes utilizados:

```
import java.io.*;
import java.lang.*;
import javax.mail.*;
import javax.mail.internet.*;
import java.util.*;
import javax.activation.DataHandler.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
```

7.1 Operaciones sobre mensajes

El método ***conecta*** efectúa la conexión al servidor de correo entrante, los parámetros necesarios para ello son tres; el nombre del servidor, nombre del usuario y contraseña.
Este método es invocado desde otra clase, devuelve *true* si la conexión ha sido establecida, en caso contrario se envía *false.*

Podemos observar que es creado un objeto *Properties,* se asocia con la sesión que se inicia al establecer la conexión.

El objeto *Properties* (propiedades) que inicializa la sesión contiene valores por defecto e información de configuración(servidor de correo, nombre de usuario y contraseña entro otros). El objeto *Session*, administra las opciones de configuración y utiliza los datos de autorización(nombre de usuario y contraseña) para interactuar con los sistemas de mensajes.

Los clientes logran acceder al almacenamiento de mensajes (*Message Store*) mediante la obtención de un objeto Store(almacén) que implementa el protocolo de acceso a la base de datos, en este caso el protocolo indicado es *POP3.*

```
public static boolean conecta(String servidor, String usuario, String
contrasena)
{
    try {
        Properties props = new Properties();
        Session session = Session.getDefaultInstance(props, null);
        store = session.getStore("pop3");
        store.connect(servidor, usuario, contrasena);
        return(true);
        }
        catch (Exception ex) {
        System.out.println("-----ERRROR AL ESTABLECER LA CONEXIÓN");
        return(false);  }
}
```

Después de que la conexión se ha establecido el siguiente paso es abrir el buzón de correo, como se describió anteriormente cada buzón contiene carpetas, en las cuales están contenidos los mensajes. La carpeta –Bandeja de Entrada- *(INBOX*) contiene todos los mensajes, por lo cual al abrirla aseguramos el acceso a cada mensaje contenido. La sintaxis se indica en el siguiente fragmento de código:

```
1    try {
2        folder = store.getFolder("INBOX");
3         try {
4             folder.open(Folder.READ_WRITE);
5            }catch (MessagingException ex) {
6             System.out.println("-----ERROR AL ABRIR EL FOLDER");
7             folder.open(Folder.READ_ONLY);
8            }
9           int totalMessages = folder.getMessageCount();
10          if (totalMessages == 0) {
11                JOptionPane.showMessageDialog(null,
12                "No hay mensajes en el buzón de correo electrónico .",
13                "Información",
14                JOptionPane.INFORMATION_MESSAGE);
15             folder.close(false);
16             store.close();
17        }
18        else{
18           Message[] msgs = folder.getMessages();
19           for (int i = 0; i < msgs.length; i++) {
20                atributos_mensaje(msgs[i],crea_lista,true,false);
21           }
22        }
```

Podemos observar que todo el contenido de la carpeta *INBOX* es asignado a la variable folder cuyo tipo es *Folder*(línea 2), posteriormente se especifica el modo de apertura, hay dos opciones: lectura-escritura(*READ_WRITE*) o sólo lectura (*READ_ONLY*); el modo solo lectura o permite no permite la operación borrar mensajes, para el sistema desarrollado

dicha operación es requerida, por lo cual la carpeta(fólder) se abre en modo lectura-escritura.
En el código anterior también se indica la operación para determinar si hay mensajes en el buzón(línea 9), si el número de mensajes es igual a cero, aparecerá una cuadro de diálogo indicando que el buzón está vacío. En las líneas 15 y 16 se especifica el procedimiento para cerrar tanto el fólder como la sesión.

Si el número de mensajes es distinto de cero, se inicia el proceso para obtener los atributos de cada mensaje.

En la línea número 2, el contenido de la carpeta *INBOX* es asignado a una variable tipo *Folder*, en la línea 18 el contenido de la variable folder es asignado a un arreglo(*msgs*) de tipo *message*, de esta manera es posible acceder a cada mensaje indicando algún índice en la variable *msgs*.

El siguiente paso consiste en efectuar operaciones sobre los mensajes y obtener sus atributos

```
23 public static void atributos_mensaje(Message m …) throws Exception {
24 String asunto,fecha;
25 String aux;
26 int numero;
27 String msg_id = new String();
28 Address[] dir;
29 String remitente;

30 if ((dir = m.getRecipients(Message.RecipientType.TO)) != null) {
31         for (int j = 0; j < dir.length; j++)
32         System.out.println("Para+:  " + dir[j].toString());
33 }
34 if ((dir = m.getRecipients(Message.RecipientType.CC)) != null) {
35         for (int j = 0; j < dir.length; j++)
36          System.out.println("Con Copia para...+:  " +
dir[j].toString());
37 }
38 if ((dir = m.getRecipients(Message.RecipientType.BCC)) != null) {
39    for (int j = 0; j < dir.length; j++)
40    System.out.println("Con Copia Oculta para...+:  "
+dir[j].toString());
41 }
42    //remitente
43    if ((dir = m.getFrom()) != null)
44    // asunto
45    asunto = m.getSubject();
46    // fecha
47    Date d = m.getSentDate();
48    if( d!= null){
49      fecha = d.toString();
50    }
51    else{
52   fecha = "Desconocida";
.    }
.
```

Mediante el fragmento de código anterior(líneas 23-52) se obtienen los atributos de cada mensaje, los cuales son: dirección de correo electrónico del remitente, destinatarios, asunto, y la fecha en que ha sido recibido el mensaje.

El método *atributos_mensaje(Message m ...)* es invocado para cada mensaje contenido en el buzón a través del ciclo *for* de las líneas 19,20 y 21.

El código de las líneas 30-41 obtiene los parámetros de las opciones para(*to*), con copia para(*CC*) y con copia oculta para(*BCC*) de cada mensaje, para ello se utiliza el método *getRecipients(tipo de recipiente)*, donde tipo de recipiente corresponde a los parámetros *TO, CC, BCC*; para recibir dichos valores se utiliza un arreglo de variables tipo *Address,* posteriormente son convertidas en tipo cadena(*string)* con el método *toString()*.

El siguiente código obtiene las banderas de cada mensaje, las cuales se utilizan para saber su estado: leído, contestado, e incompleto principalmente. Es importante mencionar que la *API JavaMail* está diseñada con base en los estándares de correo electrónico para sistemas abiertos; el protocolo *POP3* sólo define la bandera *DELETED*. La especificación del protocolo *IMAP* sí define el resto de las banderas

```
53 Flags flags = m.getFlags();
54    StringBuffer sb = new StringBuffer();
55    Flags.Flag[] sf = flags.getSystemFlags();
56    boolean first = true;
57    for (int i = 0; i < sf.length; i++) {
58        String s;
59        Flags.Flag f = sf[i];
60        if (f == Flags.Flag.ANSWERED)
61              s = "\\Constestado";
62        else if (f == Flags.Flag.DELETED)
63          s = "\\Borrado";
64        else if (f == Flags.Flag.DRAFT)
65          s = "\\Incompleto";
66          else if (f == Flags.Flag.RECENT)
67          s = "\\Reciente";
68        else if ((f == Flags.Flag.SEEN))
69          s = "\\NO ha sido leído";
70        else
71          continue;
72        if (first)
73          first = false;
74        else
75          sb.append(' ');
76        sb.append(s);
77    }
. . . . .
```

En la línea 53 se obtienen las banderas para el mensaje m, el cual es el parámetro recibido por el método *atributos_mensaje()*; posteriormente en la línea 55 se obtiene un arreglo de tipo *Flags* (tipo de variable definida en la *API JavaMail*) cuyo contenido son las banderas, para determinar cual bandera está activada se utiliza el ciclo *for* (líneas 57-77); el procedimiento simplemente consiste en acceder a cada elemento del arreglo *sf*, preguntar a que tipo de bandera corresponde el valor almacenado en el, y con base en el valor almacenado(*true o false)*, indicar si dicha bandera está activada(si el valor es *true* la bandera está activada).

Cuando se obtienen los datos de cada bandera es posible indicarle al usuario el estado de los mensajes almacenados en su buzón de correo.

Eliminar mensajes es una operación que utiliza la bandera *DELETED*, el siguiente fragmento de código elimina un mensaje.

```
78 public static void elimina_mensaje( int indice ) throws Exception {
79 Message me = null;
80        Message message[] = folder.getMessages();
81        folder.setFlags(message, new Flags(Flags.Flag.DELETED), false);
82        message[indice].setFlag(Flags.Flag.DELETED,true);
83        folder.close(true);
.....
```

El procedimiento para eliminar mensajes consiste en activar la bandera de borrado (poner con valor *true* la bandera *DELETED*), posteriormente cerrar el fólder con el argumento *true*, para que los cambios en las banderas se efectúen, de lo contrario no serán eliminados los mensajes. Todo este proceso se indica en las líneas de código 78-83; este código también muestra la forma en la cual es posible obtener el objeto mensaje a partir de su número(cada mensaje contenido en le buzón tiene asociado un número, el cual se actualiza cada vez que hay cambios en el fólder).

Previo a que finalice la aplicación se debe cerrar el fólder y la conexión, para ello simplemente se utilizan las siguientes instrucciones:

```
public static void cierra_folder() throws Exception{

folder.close(true);
}

public static void cierra_conexion() throws Exception{
store.close();
}
```

7.2 Envío de mensajes

En el capítulo 6 se mencionó que en la interfaz para envío de mensajes era posible adjuntar archivos de cualquier tipo; pero el cuerpo del mensaje únicamente podría contener texto simple.

El envío de mensajes es una operación que puede dividirse en cuatro etapas; especificar datos de encabezado(remitente, destinatarios y asunto), escribir el mensaje, adjuntar archivos y enviarlo.

El siguiente fragmento de código efectúa la primera y la segunda tarea, además especifica cual será el servidor de correo saliente:

```
1  String host = "smtp.prodigy.net.mx"; //servidor de correo saliente
2  String subject="hola";
3  Properties props = System.getProperties();
4  props.put("mail.smtp.host", host);
5  Session session = Session.getInstance(props, null);


6         MsgText1= "Mensaje de prueba" //cuerpo del mensaje.
7     try {
8         MimeMessage msg = new MimeMessage(session);
9         msg.setFrom(new InternetAddress(from));

10        if( a1 != null )
11        msg.setRecipients(Message.RecipientType.TO,
                            InternetAddress.parse(a1,false));
12        if( to1 != null )
13        msg.setRecipients(Message.RecipientType.CC,
14                          InternetAddress.parse(to1,false));

15        if( to != null )
16        msg.setRecipients(Message.RecipientType.BCC,
17                          InternetAddress.parse(to,false));

18        msg.setSubject(subject);

19        MimeBodyPart mbp1 = new MimeBodyPart();
20        mbp1.setText(msgText1);
.....
```

Las líneas 3-5 se utilizan para especificar el servidor de correo saliente a utilizar durante las operaciones de envío, el cual corresponderá al servidor de correo saliente del *ISP* del cliente que ejecute la aplicación.

Las líneas 8-18 definen los datos de encabezado. En la línea 8 se instancia un objeto de tipo *MimeMessage* utilizado para especificar direcciones de correo electrónico del

remitente(línea 9), destinatarios(líneas 11 y 12) , copias(línea 13 y 14), copias ocultas(16 y 17) y asunto(línea 18).
En las líneas 19 y 20 se describe la segunda tarea; en la línea 19 se instancia un objeto *MimeBodyPart* llamado *mbp1,* el cual le es asignado el contenido del mensaje(línea 20).

En el capítulo 4(*API JavaMail*) se mencionó que cada mensaje contiene dos secciones: atributos de encabezado y el cuerpo de contenido. El cuerpo de contenido puede estar integrado por uno o más objetos llamados *MimeBodyPart*; cada uno de éstos corresponde a una sección del mensaje, por lo cual tanto el mensaje que el usuario escriba en la interfaz como cada archivo que adjunte serán colocados en un objeto multiparte (*Multipart*), finalmente el objeto multiparte será colocado como contenido del mensaje.

Las líneas 21 y 22 añaden el objeto *MimeBodyPart* correspondiente al cuerpo del mensaje(líneas 19 y 20) al objeto multiparte.

El procedimiento para adjuntar archivos está descrito en las líneas 23-45.

```
21:    Multipart mp = new MimeMultipart();
22:    mp.addBodyPart(mbp1);

23:    MimeBodyPart adjunto1;
24:    FileDataSource archivo1;
25:    String nuevo;

26:    Enumeration enum = datos.elements();

27:       while ( enum.hasMoreElements() ){
28:               nuevo = enum.nextElement().toString();
29:               adjunto1 = crea();
30:               archivo1 = crea1(nuevo);

31:               adjunto1.setDataHandler( new DataHandler(archivo1));
32:               adjunto1.setFileName(archivo1.getName());

33:               mp.addBodyPart(adjunto1);
34:      }

35:       msg.setContent(mp);
36:       msg.setSentDate(new Date());

37:       Transport.send(msg);
.....
38:   public MimeBodyPart crea(){
39:     MimeBodyPart adjunto = new MimeBodyPart();
40:     return(adjunto);
41:   }

42:   public FileDataSource crea1(String nom){
43:     FileDataSource archivo  = new FileDataSource(nom);
44:     return(archivo);
45:   }
```

Se declaran tres variables: adjunto1 de tipo *MimeBodyPart*, archivo1 de tipo *FileDataSource*, y nuevo de tipo cadena(líneas 23-25).

En la línea 26 a la variable enum de tipo *Enumeration* le es asignada la lista con el nombre de todos los archivos adjuntos, en dicha lista se ha comprobado previamente que el tamaño de los archivos no es superior a 3 *MB*.

El número de interacciones del ciclo *while* corresponderá al número de archivos en la lista (esta lista de archivos se genera cuando el usuario especifica a través de la interfaz cuales mensajes adjunta). En cada iteración se accede a un distinto archivo para colocar su contenido en un objeto *MimeBodypart*.

El proceso para colocar cada archivo en un objeto *MimeBodyPart* se lleva a cabo en cinco pasos:

1. Hacer una instancia de tipo *MimeBodyPart* (línea 29).
2. Hacer una instancia de tipo *FileDataSource* (línea 30).
3. Colocar el contenido del archivo en un obejto *MimeBodyPart* (línea 31).
4. Especificar el nombre del archivo(línea 32).
5. Añadir dicho archivo al objeto multiparte.

Finalmente cuando han sido colocados todos los elementos que forman parte del mensaje, se especifica el objeto multiparte como contenido del mensaje(línea 35), también pueden especificarse otros parámetros, en la línea 36 se agrega la fecha de envío.

En la línea 37 se ejecuta el envío del mensaje.

8. Instalación y Pruebas

8.1 Procedimiento de instalación

Requerimientos mínimos de *hardware*:

- Computadora con microprocesador 486 a 66 *Mhz*, 16 *MB* de memoria *RAM*, disco duro de 1*GB*, y un módem 28,800 *bps*(bits por segundo) y línea telefónica. Se recomienda un equipo superior a lo que se ha descrito, ya que la ejecución del sistema podría ser demasiado lenta.

Requerimientos de *software*:

- Sistema operativo Windows 95 o LINUX(se deberá compilar nuevamente los archivos *.java del sistema), JDK versión 1.4.01 o superior. No es requerido algún navegador para Internet(Internet Explorer o Netscape), aunque este se incluye con el sistema operativo o se obtiene a través de Internet de manera gratuita.

Clases de la *API JavaMail*: se obtienen de manera gratuita en la página de *Sun Microsystems:*

http://java.sun.com/products/javamail/javamail-1_3.html

Colocar las clases *imap.jar, mailapi.jar, pop3.jar, y smtp.jar* en el directorio:

Java_home\jre\lib\ext

***Java Activations FrameWork*:** se obtiene de manera gratuita en la página de *Sun Microsystems;*

http://java.sun.com/beans/glasgow/jaf.html.

Colocar el archivo *activation.jar* en el directorio:

Java_home\jre\lib\ext

Después de que las clases anteriores han sido colocadas en los directorios, se ejecuta lo siguiente bajo *Windows*:

Colocar en un directorio la carpeta sistema (contiene tanto las clases de java como los archivos *.class* del sistema).

java inicio (para ejecutar el sistema).

8.2 Pruebas

Las pruebas se llevaron a cabo en una máquina con las siguientes características; procesador *Pentium III*, memoria *RAM* de 524 *MB*, sistema operativo Windows 2000.

Para ejecutar la aplicación se utiliza la siguiente instrucción: (desde una ventana de sistema).

java inicio

con lo cual se obtiene la ventana –Registro- (primera imagen de la figura), si los datos ingresados por el usuario son correctos, se desplegará la ventana Menú_Principal, la cual muestra cuatro botones(correspondientes a las opciones *Enviar, Diagnostico, Responder y Filtro*.) y una barra de menú. Figura 8.1

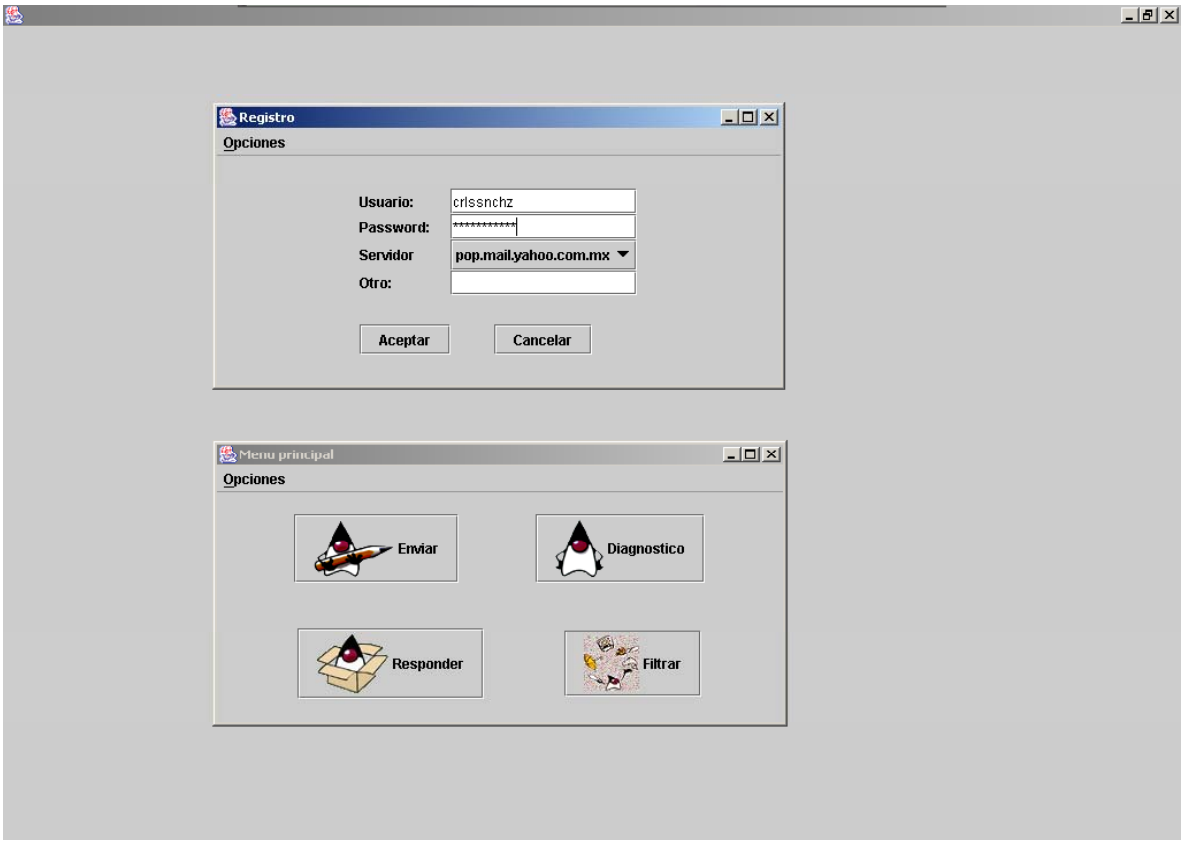


Figura 8-1. Ventana de registro y menú principal

Si el usuario selecciona la opción Enviar, obtendrá la siguiente pantalla:

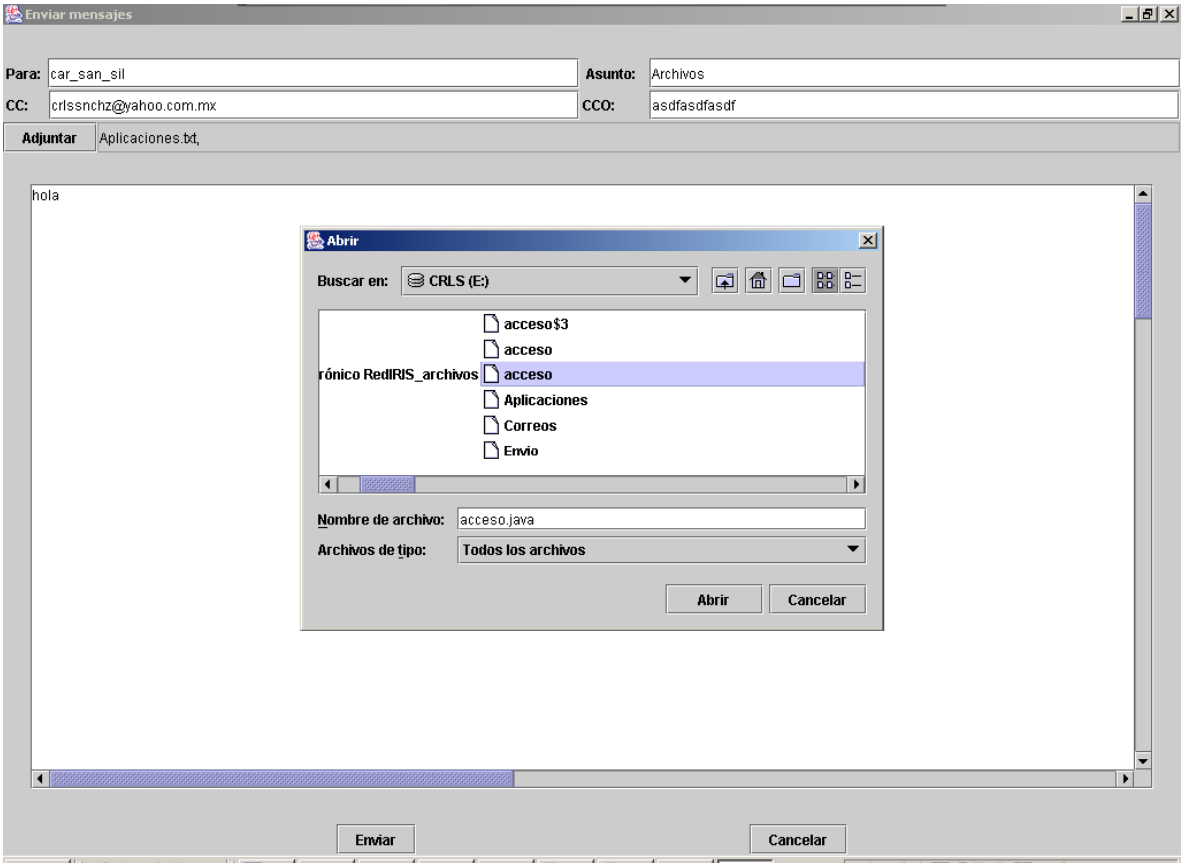


Figura 8.2 Opción Enviar

Podemos observar en la figura 8.2 la interfaz para envío de mensajes, la cual contiene los campos *Para, Asunto, CC, CCO, el área de texto y los botones Enviar, Cancelar y Adjuntar* .

Cuando el usuario hace clic sobre el botón *Adjuntar,* se despliega el cuadro de diálogo *Abrir* para que el usuario seleccione el archivo que desea adjuntar. El usuario puede cancelar la operación de envío, simplemente debe oprimir el botón *Cancelar*.

Cuando el usuario ha enviado el mensaje se desplegará un cuadro de diálogo, para indicar si ha ocurrido algún error durante el envío. La figura 8.2, muestra que los campos de texto *Para y CC* han sido llenados con direcciones no válidas de correo electrónico, por lo cual el la aplicación desplegará la pantalla de la figura 8.3 indicando el error.

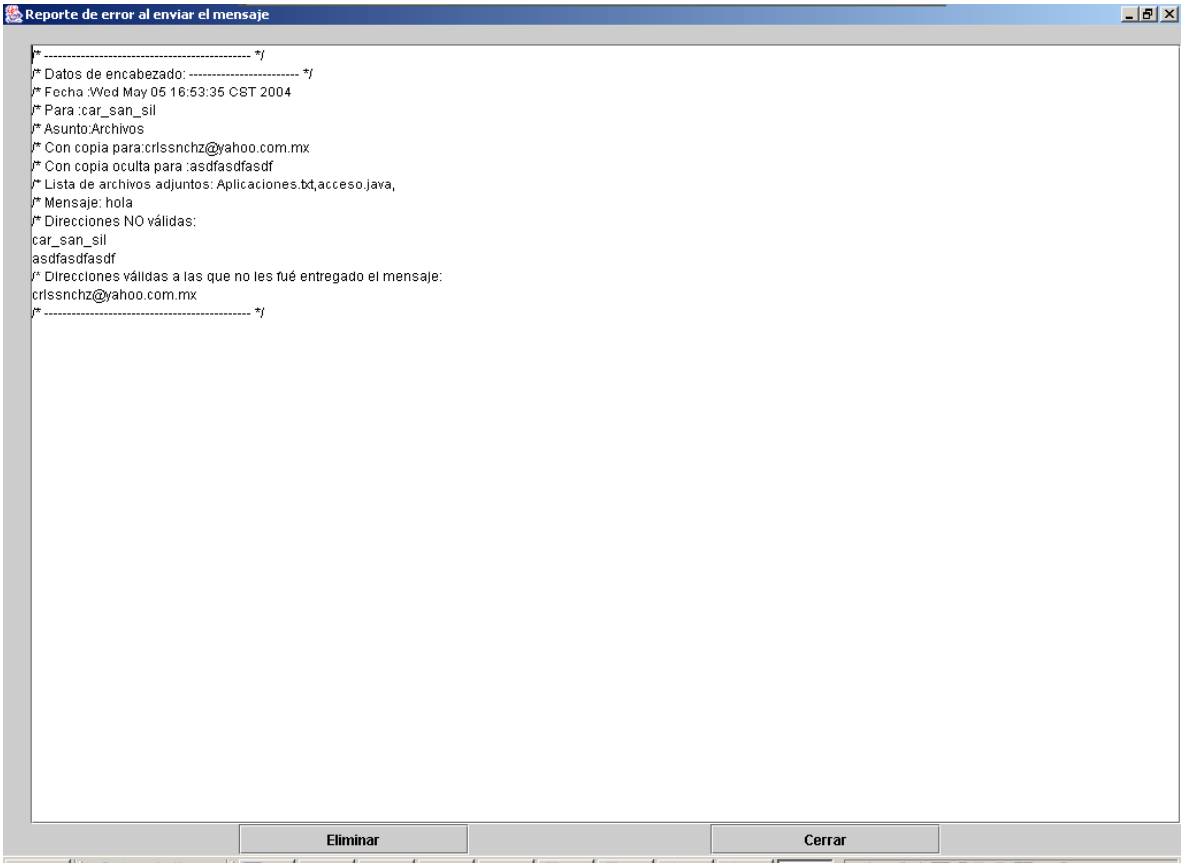


Figura 8.3 Reporte de error al enviar el mensaje

Si el usuario selecciona la opción diagnóstico aparecerá la pantalla de la figura 8.4, en la cual podemos observar el listado de los mensajes cuyos datos de remitente podrían ser no validos; también se muestran dos botones *Eliminar y Ver detalle,* y la barra de menú *Opciones(Salir, Eliminar Todos y Acerca del Sistema)* .

Para leer la línea *Received* de algún mensaje el usuario deberá seleccionarlo y oprimir el botón *Ver detalle,* con lo cual se desplegará una pantalla como la mostrada en la figura 8.4, en esta aparecen algunos de los atributos del mensaje(asunto, remitente, dominio, destinatarios y línea *Received*).

Al seleccionar la opción *Filtro* el usuario obtendrá un listado de todos los mensajes existentes en el buzón de correo, de esta manera él decidirá que filtros aplicar; cada resultado obtenido será mostrado a través de una interfaz gráfica como la que se muestra en la figura 8.5, en la cual podemos observar las opciones disponibles de filtrado.

La figura 8.6 muestra el resultado obtenido al aplicar los filtros especificados en la figura 8.5.

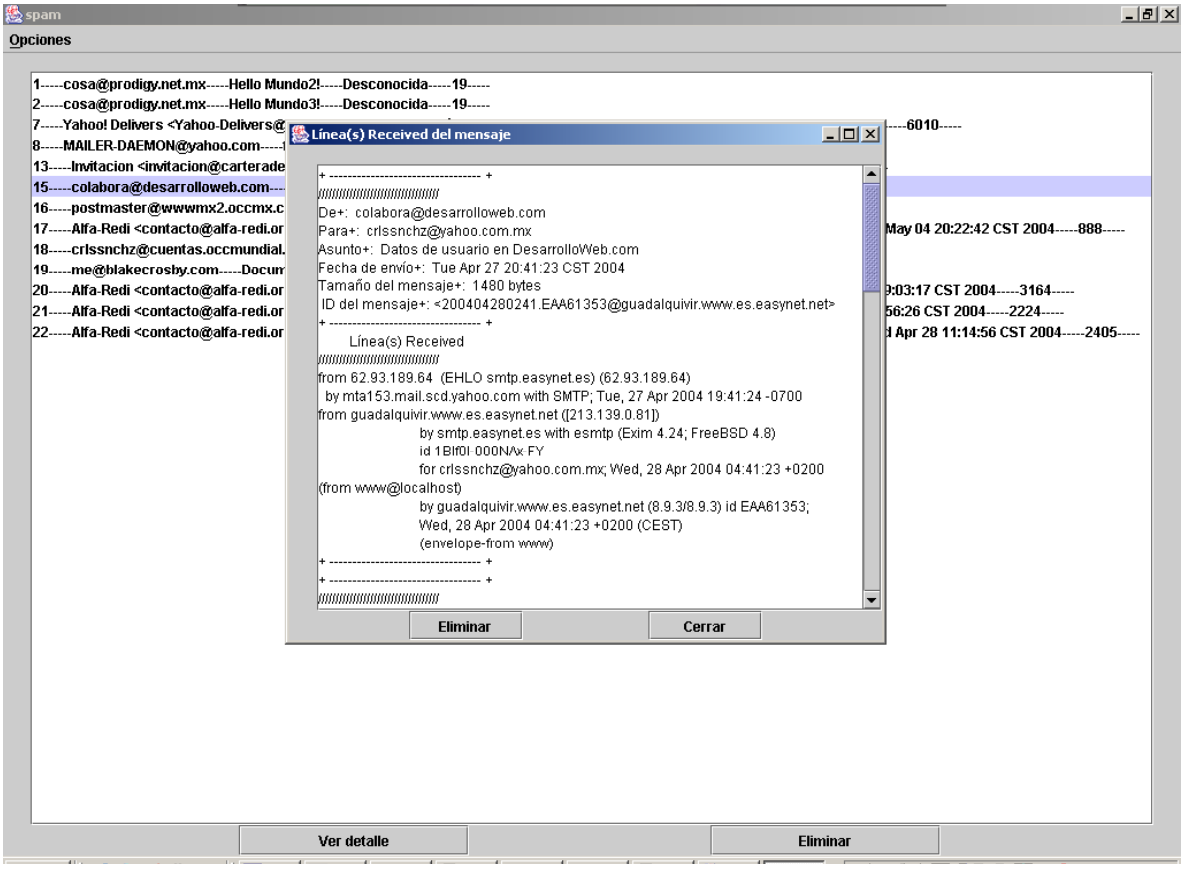


Figura 8.4 Listado de mensajes cuyos datos de remitente podrían ser falsos

Podemos observar en la figura 8.5 las siguientes opciones de filtrado:

1. *Asunto: [boletín-ar] Alfa-Redi: Taller sobre Gobernabilidad del Internet-Peru*
2. Remitente: crlspppIcih@prodigy.net.mx
3. Dominio: deitel.com
4. Considerar: alguna condición.

La opción 4, de filtrado está especifica con el argumento *alguna condición,* por lo que serán buscados todos los mensajes de correos que cumplan con al menos una de las condiciones.

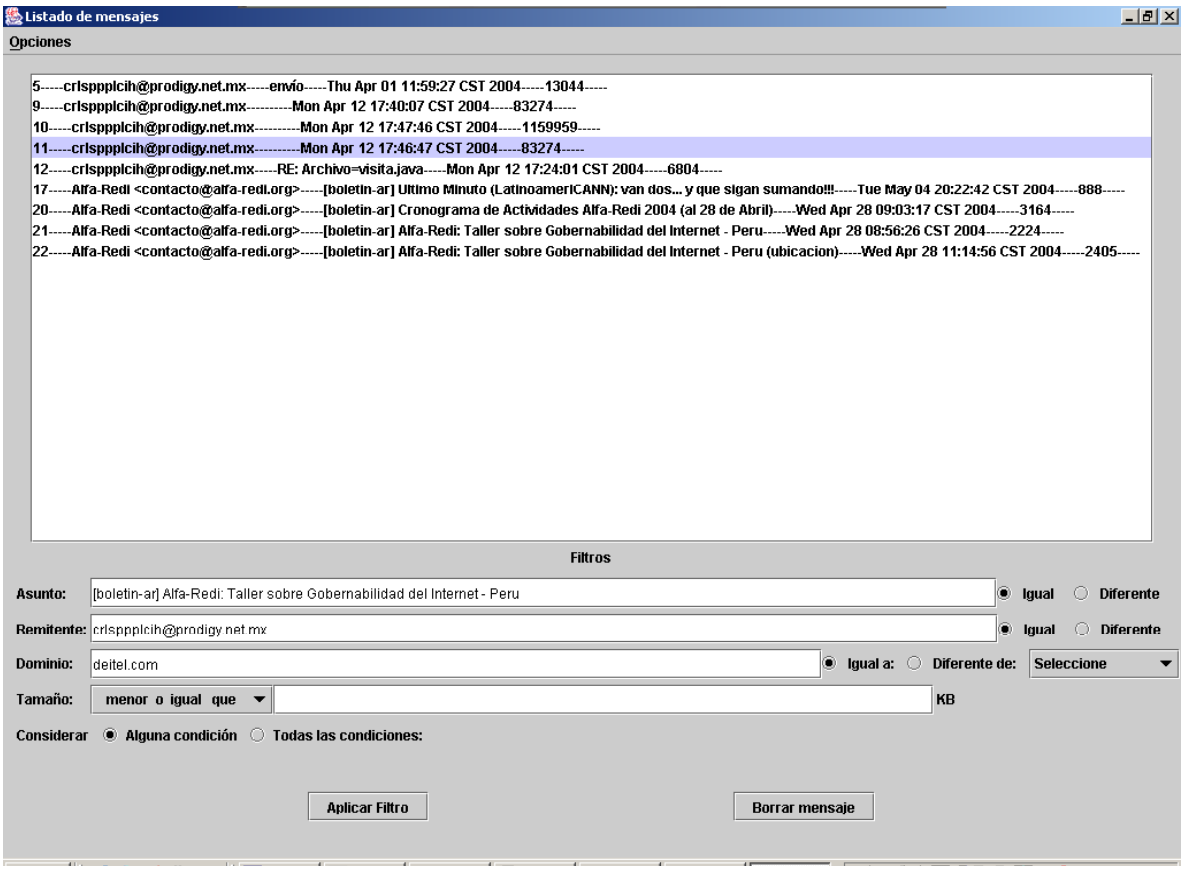


Figura 8.5 Listado de mensajes, opción Filtrar

En la siguiente figura(8.6) se muestra el resultado obtenido al aplicar los filtros indicados anteriormente.

Se observan 9 líneas, cada una indica atributos de los mensajes(número del mensaje, remitente, asunto, fecha de recepción y tamaño en *bytes*).

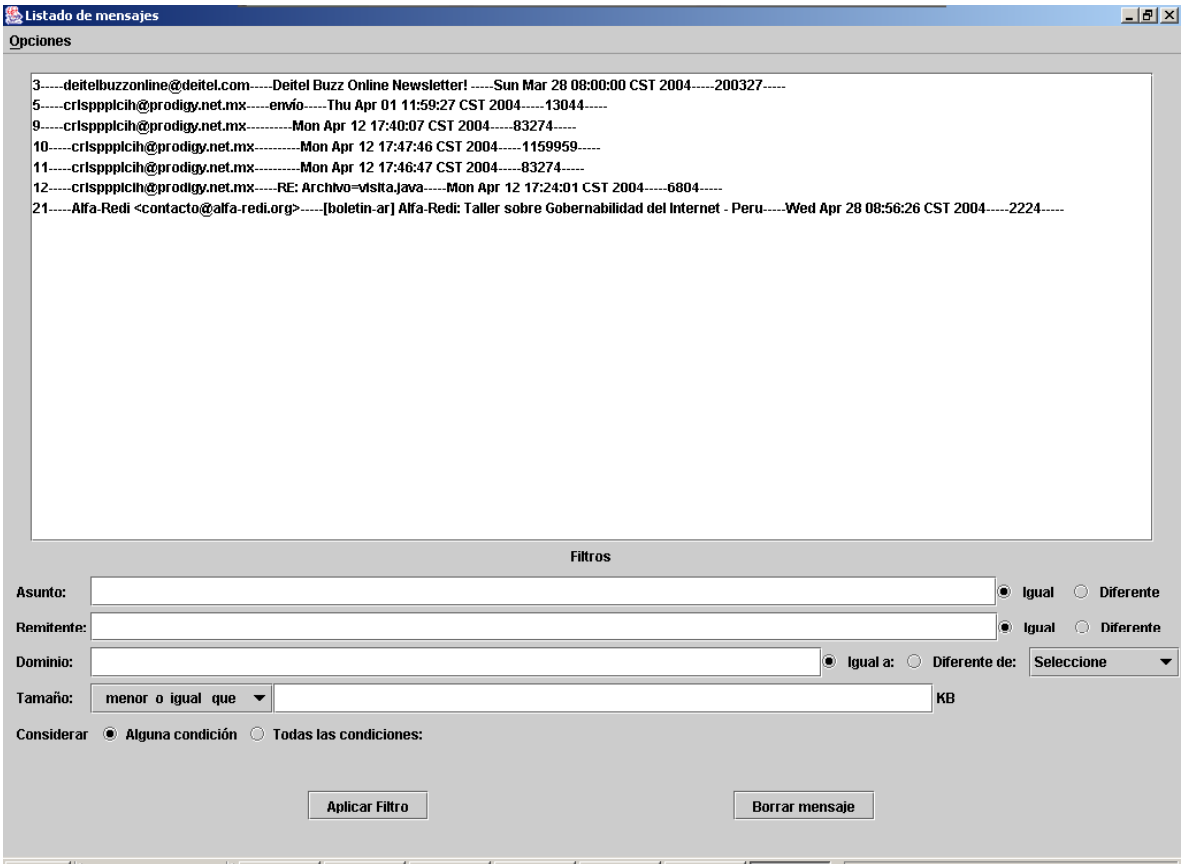


Figura 8.6 Resultado de la opción Aplicar Filtro

Finalmente, la opción *Responder* simplemente le solicita al usuario la ruta absoluta del directorio donde están almacenados los archivos que podrían ser solicitados, para que el sistema lleve a cabo la búsqueda y el envío de los mismos. El resultado de este procesamiento se muestra en la figura 8.7.

Podemos observar en la siguiente figura (8.7) el resultado generado por el sistema en respuesta al seleccionar la opción *Enviar*.

Había cinco peticiones de las cuales una de ellas solicita el archivo datos, el cual no fue encontrado, ya que en la ventana de resultados se indica

ARCHIVO(S) NO ENCONTRADO.

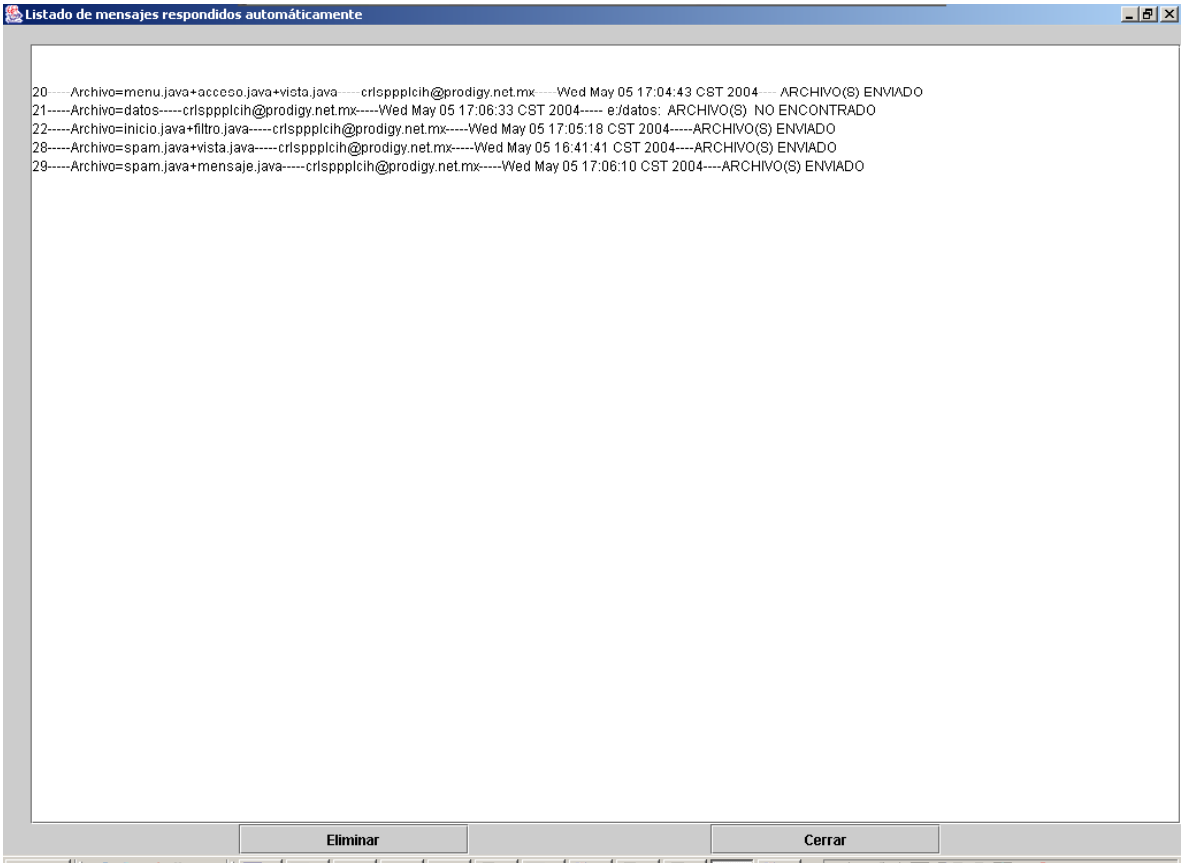


Figura 8.7 Resultado de la opción Envío.

9. Conclusión

El sistema desarrollado efectúa cada tarea descrita en la etapa de análisis; no obstante, las herramientas de desarrollo utilizadas proveen amplios recursos que podrían mejorarlo significativamente.

La *API JavaMail* se apega completamente a las especificaciones de los estándares actuales para protocolos de correo electrónico, por lo tanto es posible añadir funcionalidades que estén definidas en las especificaciones de los protocolos *POP3* e *IMAP*, ya que las clases utilizadas para este desarrollo representan un subconjunto muy pequeño de las mismas. En la *API* de cada protocolo se describe el universo de clases y métodos, los cuales es posible implementar para agregarlos en el sistema.

Considerando requerimientos mínimos para un sistema cliente de correo electrónico, el sistema desarrollado no corresponde a ello, ya que falta la implementación para mostrar el contenido de los mensajes; sí es posible acceder a este y obtener cada archivo adjunto, sin embargo tanto el lenguaje *Java* como la *API JavaSwing* no proveen el medio para visualizar contenido *HTML*. Si el sistema muestra en pantalla el mensaje, resultaría inadecuado, al usuario le llevaría tiempo entender completamente el contenido, ya que observaría los datos y las etiquetas propias del lenguaje *HTML*.

El sistema carece de una base de datos que permita definir un esquema para perfiles de usuarios, esta limitación es tangible cuando se utiliza la interfaz para envío de mensajes, ya que no es posible proporcionar al usuario la opción que le permita guardar direcciones de correo electrónico.

Figura 6.3 Diagrama de clases

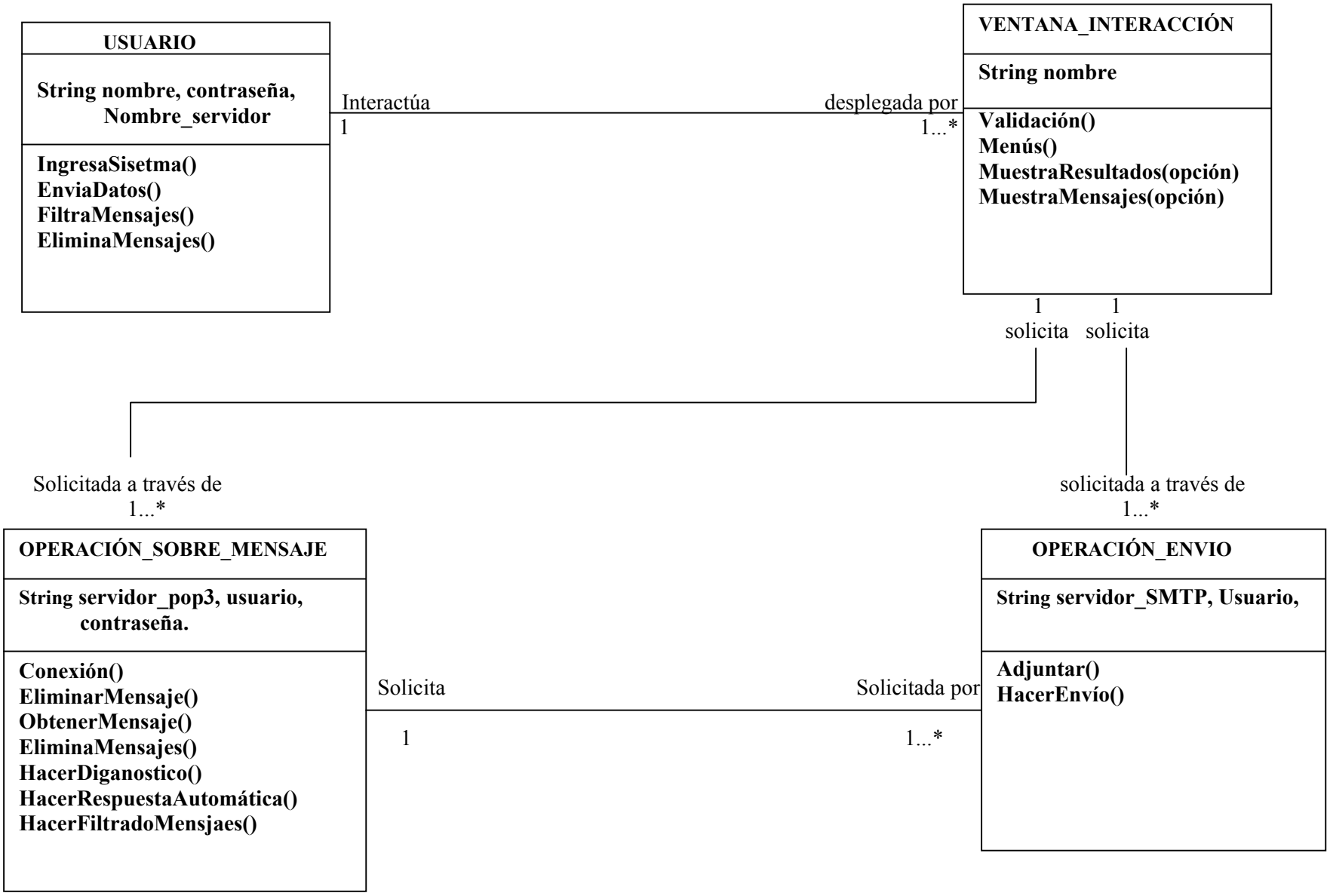


Figura 6.31 Diagrama de estado