Universidad de la República Oriental del Uruguay

Facultad de Ingeniería

Carrera Ingeniería en Computación

Sistema MailDB

Integración de e-mail y bases de datos

Integrantes

Elohim Olivera
Marcelo Rodríguez

Responsables

Ing. Felipe Zipitría
Ing. Alejandro Blanco

Proyecto de grado
Abril 2005

Resumen

Este trabajo se basa en el estudio de formatos de almacenamientos de correo electrónico (e-mail). La principal motivación de este estudio es la sobrecarga de procesamiento que sufren los servidores de e-mails al trabajar con casillas (mailboxes) de gran tamaño.

Por otro lado, es indudable el avance y potencia que han desarrollado las bases de datos, en consecuencia surgen naturalmente preguntas como:

- ¿Alguna de las soluciones de almacenamiento existentes está aprovechando ese conocimiento?
- ¿Se obtienen ventajas sobre los otros formatos de almacenamiento al utilizar bases de datos?

En la primera parte del trabajo, se estudian los formatos y tecnologías más comúnmente utilizadas hoy en día para el almacenamiento de e-mails. Mediante un estudio cuantitativo se analiza que impacto tiene en el desempeño de los servicios de e-mails, la utilización de cada formato.

En busca de mejoras en la calidad de los servicios, se analiza y diseña una API (Application Programming Interface, traducido del inglés: Interfaz de Programación de Aplicaciones) que se encarga de brindar las herramientas necesarias para trabajar con e-mails almacenados en una base de datos. Se desarrolla un prototipo de esta API y se la integra a un servidor de e-mails.

Por último, se compara el desempeño de los servicios de e-mail utilizando la API con los datos obtenidos de servicios que utilizan otro tipo de formatos para almacenar e-mails.

Palabras Clave: Servicio de correo electrónico, e-mail, mailbox, formato de almacenamiento de correo electrónico, base de datos, API, servidor de correo.

Tabla de Contenido

Índice de figuras

Índice de tablas

Capítulo I

Introducción

Motivación

El avance tecnológico y su popularización han hecho del e-mail un método de comunicación práctico, ágil, cómodo y por ende, cada vez más preferido en los distintos ámbitos (académico, comercial y doméstico).

El concepto de e-mail ha ido evolucionando. Hoy en día, no se trata solamente de enviar texto, sino que también se pude adjuntar archivos de distintos formatos, por ejemplo: sonido y video que son codificados en texto[1], de esta manera, se producen e-mails de tamaño considerable.
Los mecanismos para acceder a esta tecnología están cada vez más disponibles; es cada vez más común el ofrecimiento de cuentas de e-mail gratuitas.

Los factores anteriormente citados y otros, como ser el envío de e-mail basura (spam) [2], ponen a prueba constantemente la calidad del servicio ofrecida por los servidores de e-mail. Más aún, implica una creciente y cada vez más importante, demanda de recursos disponibles para mantener la calidad de servicio. Naturalmente, esto tiene asociado ciertos costos, por ejemplo, mayor espacio en disco, mayor ancho de banda, sobrecarga de los sistemas, etc.

El avance del hardware y la disminución de sus costos junto al adelanto tecnológico de los discos, buses, memorias y telecomunicaciones actúan como un moderador de la problemática.
Pero en muchas instituciones, principalmente de nuestro país, no siempre se cuenta con el presupuesto necesario para proveer a sus servidores de la última tecnología disponible. Surgen entonces diferentes alternativas, como lo pueden ser la distribución de cargas y tareas entre equipamiento de menor porte, y por lo tanto mas económicos. Si bien esto surge como una alternativa viable, genera mayor complejidad en lo que se refiere a la infraestructura de los servicios, acarreando consigo mayores complejidades de administración.

Uno de los factores que es afectado directamente por el crecimiento masivo del e-mail es el mecanismo de almacenamiento subyacente.
Este proyecto pone en tela de juicio las técnicas y soluciones existentes en lo que respecta al almacenamiento de e-mails. Uno de los objetivos es entonces determinar si la aplicación de la tecnología y la potencia ofrecida por bases de datos, como formato de almacenamiento, permite brindar un mejor servicio.

Viabilidad

Existen trabajos realizados en la dirección de almacenar los e-mails en bases de datos (tanto de distribución libre [3], [4] como propietaria [5]).

El enfoque que siguen estos trabajos es implementar sus propios servidores de correo entrante y saliente. Este proyecto apunta a otra dirección, que consiste en brindar una API (Application Programming Interface) que maneje el almacenamiento de e-mails en un base de datos y sirva de ayuda a los programadores de servidores.

Una carencia de los productos encontrados es que en ninguno de los casos se encontró documentación que justifique las mejoras proporcionadas al incluir bases de datos como formato de almacenamiento de e-mails.

Objetivos

Analizar las tecnologías referentes a los formatos de almacenamiento de e-mails, en particular el almacenamiento en bases de datos; identificando las ventajas y desventajas de cada uno de ellos. Realizar un análisis de desempeño de los diferentes formatos investigados, para ello se hará uso de diferentes aplicaciones que implementen servidores de e-mail y utilicen dichos formatos.

Desarrollar un sistema que sea capaz de manejar el almacenamiento de e-mail en una base de datos. Ofrecer una API pública que permita con un mínimo esfuerzo ser aplicada a diferentes implementaciones de servidores de e-mails.
Integrar este nuevo sistema a alguna de las implementaciones de servidores analizadas y someterla a los mismos chequeos de desempeño que se sometieron los otros formatos.

Organización del documento

El presente documento está integrado por 6 capítulos. En el capítulo 2 se introducen los conceptos básicos del funcionamiento de un sistema de e-mails y a continuación se presenta una revisión del estado del arte en lo que refiere a los diferentes formatos de almacenamiento.

En el capítulo 3 se investiga cuál es la realidad en la que está inmerso el problema, para seguidamente realizar el planteamiento de la solución.

En el capítulo 4 se presenta el prototipo implementado en este proyecto.

En el capítulo 5 se realiza un análisis comparativo entre los diferentes formatos de almacenamiento, incluido el formato basado en bases de datos propuesto como solución.

Para finalizar, en el capítulo 6 se presentan las conclusiones obtenidas en el transcurso del proyecto junto a un planteo de posibles trabajos futuros.

Capítulo II

Estudio de tecnologías existentes.

Resumen

En este capítulo, se presenta una revisión del estado del arte en lo que refiere a los diferentes formatos de almacenamiento de e-mail. Se estudian también productos que implementan el almacenamiento de e-mails en dichos formatos.

Para facilitar la comprensión de los temas tratados en este capítulo y en los siguientes, se brinda a continuación una breve descripción del funcionamiento de un servicio de e-mails y de los diferentes agentes que participan en esta tarea.

La próxima sección puede ser obviada por el lector, si éste posee los conocimientos necesarios respecto a los temas involucrados en la recepción y envío de e-mails, pasando directamente a las siguientes secciones del presente capítulo.

Fundamentos del sistema de e-mails

El marco de trabajo (*framework*) que hace posible la transmisión de e-mails entre usuarios de Internet, que utilizan diferentes plataformas y tecnologías, está integrado por una colección de componentes que cumplen ciertos estándares.

Dentro de estos componentes podemos encontrar:

- Agentes.
- Protocolos de comunicación.
- Depósitos de e-mails.

En la Figura II-1 se representan gráficamente las relaciones entre los diferentes componentes de un sistema de e-mails.

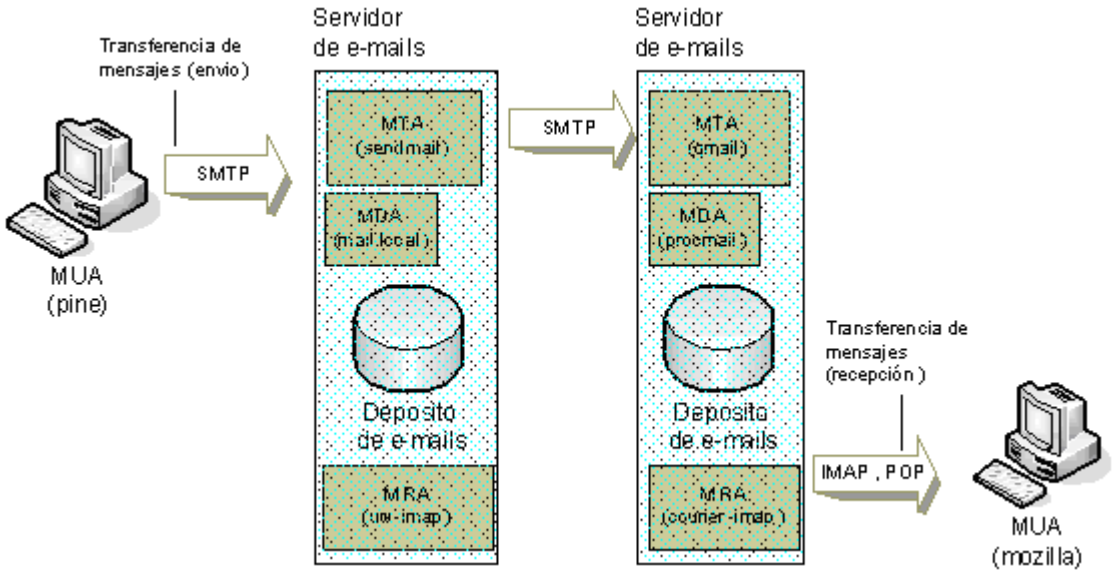


Figura II-1 Fundamentos del sistema de e-mails

Agentes

Los agentes son programas (software) que se agrupan en cuatro categorías Mail User Agent (MUA), Mail Transfer Agent (MTA), Mail Delivery Agent (MDA), y Mail Retrieval Agent (MRA). Cada uno de ellos desempeña un papel específico en el proceso de transmitir y administrar los e-mails. La mayoría de los usuaris, por lo general solamente conoce a los agentes MUA e ignora la existencia del resto.

MUA (Mail User Agent, Agente de Usuario de Correo)

Los MUA son la interfaz entre los servidores de e-mail y el usuario, le permite a éstos leer y escribir e-mails. y menudo se lo denomina *cliente de correo*. Cuando el usuario quiere enviar un e-mail utiliza el MUA para redactarlo, una vez redactado, el usuario le indica al MUA que envíe el nuevo e-mail al servidor. Acto seguido, nuestro MUA se comunica con el servidor y le pasa el e-mail que el usuario quiere enviar, utilizando en esta comunicación el protocolo SMTP.

También se utilizan los MUA para acceder los e-mails que el usuario tiene almacenados.
Los e-mails pueden estar almacenados en archivos locales a la máquina donde está ejecutando el MUA y/o en los servidores. El MUA accede a los e-mails almacenados en archivos locales por intermedio de llamadas al sistema de archivos. Para acceder a los e-mails que están almacenados en el servidor utiliza los protocolos IMAP o POP.

Tres ejemplos de MUAs son *mozilla* [6], *outlook* [7] y *pine* [8].

MTA (Mail Transfer Agent, Agente de Transferencia de Correo)

Los MTA son los programas encargados de transportar los e-mails entre servidores, hasta que éstos llegan a su destino. El MTA de envío recibe el e-mail desde el MUA del usuario y lo transmite al MTA de destino. En este proceso el MTA determina si el e-mail va con destino a un usuario local o no. Si va dirigido a un usuario local lo pasa al MDA que es quien se encargará de efectivizar la entrega, en caso contrario delega el e-mail al MTA que corresponda. Un e-mail puede pasar por varios MTAs antes de llegar a su destino final.

Ejemplos de MTAs son *sendmail* [9] y *qmail* [10].

MDA (Mail Delivery Agent, Agente de Entrega de Correo)

El MDA, también denominado LDA (Local Delivery Agent), es el responsable de conocer el formato utilizado para almacenar los e-mails y la ubicación del depósito.
Tiene como tarea principal almacenar en el depósito los e-mails que le pasa el MTA.
Hay que tener en cuenta que los agentes LDA no transportan mensajes entre sistemas ni actúan como interfaz para el usuario final.

Ejemplos de MDA conocidos son, *procmail* [11] y *mail.local* [12]

MRA (Mail Retrieval Agent, Agente de Recuperación de Correo)

El MRA es un programa que actúa como un intermediario entre el MUA del usuario y el depósito de e-mails que está en el servidor.
Al igual que el MDA, conoce el formato utilizado para almacenar los e-mails y la ubicación del depósito.
El MUA se comunica con el MRA para obtener los e-mails que están almacenados en el servidor, en esta comunicación se utiliza el protocolo IMAP IMAP (Internet Message Access Protocol) [13] o el protocolo POP (Post Office Protocol) [14]

Ejemplos de MRA conocidos son *uw-imap* [15] y *courier-imap* [16]

Protocolos de comunicación

Brevemente describiremos los estándares que intervienen en el transporte de un e-mail entre los diferentes agentes.
El protocolo SMTP (Simple Mail Transfer Protocol) [17]define como un e-mail debe de ser transportado entre agentes, este transporte puede ser desde el MUA al MTA o entre MTAs.
La definición de este protocolo se puede encontrar en detalle en el RFC 2821 [18]

Para acceder a los correos que el usuario tiene almacenados en el servidor, el MUA se comunica con el agente MRA utilizando los protocolos IMAP y POP estos protocolos están definidos en los RFC 2060 [19] y RFC 1939 [20] respectivamente.

Depósito de e-mails

El depósito de e-mails es el lugar físico donde éstos son almacenados al ingresar en el servidor. Cuando el servidor de e-mails recibe un e-mail dirigido a un usuario local del sistema, éste almacena el mismo en la porción del depósito de e-mails correspondiente al usuario. Para almacenar los e-mails se utilizan diferentes formatos, en la siguiente sección de este capítulo se estudian dichos formatos.

Ejemplos de formato de almacenamiento son *mbox* [21]y *maildir* [22].

Formatos de almacenamiento de e-mails

En general, podemos agrupar los formatos de almacenamiento de e-mails en dos categorías: basados en archivo y basados en directorios.

A continuación pasaremos a detallar estos dos enfoques junto a sus ventajas y desventajas.

Formatos basados en archivo.

En este tipo de formatos todos los e-mails son almacenados en un único archivo, se utiliza un separador dentro de este archivo para distinguir el comienzo y fin de cada e-mail.
A continuación describiremos las variantes más comunes.

Mbox

Esta es la variante más común dentro de los formatos basados en archivos, es el formato clásico de los sistemas Unix.

En esta variante del formato los e-mails están delimitados superiormente por una línea que comienza con la palabra "From" seguida de un espacio en blanco, llamaremos a ésto token superior, e inferiormente por una línea en blanco (token inferior).

Para no interferir con el formato de los delimitadores, antes de agregar un nuevo e-mail al mailbox es necesario revisar cada línea de éste. Si en el cuerpo del e-mail existe una línea que comience con el token superior se tiene que tomar alguna acción. La acción tomada por el formato *mbox* es convertir la línea agregándole delante el carácter ">".
Cuando se retira un e-mail del mailbox es necesario volver atrás la transformación, se tienen que eliminar todos los caracteres > de las líneas que tengan la forma ">token superior …..".

Esté mecanismo acarrea un inconveniente, luego de que el mensaje es convertido y agregado al mailbox, es imposible distinguir entre líneas que comenzaban con ">token superior " en el e-mail original, de líneas que comenzaban con el token superior y fueron convertidas. Claramente, cuando se retira un e-mail del mailbox éste resulta alterado, pues la tarea llevada adelante al retirar el e-mail es eliminar todos los caracteres ">" que preceden al token superior.

Otras variantes de este formato son:

Mboxrd

Es similar al *mbox*, con la diferencia de que al momento de agregar el e-mail al mailbox, se inserta un carácter ">" por cada línea que contenga cero o mas caracteres ">" seguidos del token superior. Solucionando de esta manera el inconveniente de *mbox*.

Mmdf

Es similar al *mbox* pero utiliza como token cuatro caracteres "control-A" (caracter ASCII 01). La única implementación que utiliza este formato es el Unix de SCO.

Mbx

Es una variante del formato basado en archivo, que tiene como particularidad agregar información en binario al inicio del archivo de mailbox para agilizar el acceso a los e-mails.
Soporta el acceso concurrente, los inconvenientes son que utiliza técnicas de bloqueo de archivos que no son confiables al utilizar NFS y si se corrompe el mailbox no es sencilla su recuperación (por la información binaria que este contiene).

En general la desventaja más seria del formato basado en archivos, es el acceso simultáneo al mailbox por múltiples procesos. Si los procesos acceden utilizando mecanismo de sólo lectura no se generan problemas. Los problemas comienzan cuando alguno de los procesos necesita acceso de escritura, en el caso de que más de un proceso acceda en modo escritura, el mailbox puede resultar corrupto.
Para solucionar estos inconvenientes y prevenir que el mailbox resulte corrupto, se tiene que agregar algún mecanismo de bloqueo de archivos (file locking).
Existen varios mecanismos de bloqueos de archivos, pero cuando se trabaja con sistemas de archivos compartidos (como por ejemplo NFS [26]se pueden generar algunos inconvenientes.

Otra desventaja, es la imposibilidad de acceder directamente a un e-mail, pues al estar todos agrupados en un único archivo es necesario recorrerlo (parsear) para poder identificar un mensaje en particular. De forma similar, para borrar un e-mail es necesario localizarlo, eliminarlo y reescribir todo el mailbox.

Una ventaja que puede surgir, es al momento de realizar alguna búsqueda dentro del mailbox. La razón de la ventaja viene por el lado de que se tiene que abrir un solo archivo para buscar la información, mientras que, como veremos, en los formatos basados en directorios se tiene que abrir cada uno de los archivos que representa un e-mail.

Aparte de las variantes citadas existen otras como ser mtx, dbx, phile, tenex, mboxcl, mboxcl2, por más información sobre estas variantes ver las referencias [23] y [24].

Formatos basados en directorios

En este formato los e-mails son almacenados dentro de una estructura de directorios (un directorio o más), en un único archivo por e-mail.

Pasamos a describir las variantes más comunes de este formato.

Mh

Esta variante es la más antigua dentro de este formato, es la primera en almacenar los e-mails en archivos individuales.
Almacena los archivos, e-mails, en un único directorio, el nombre asignado a cada archivo es un número. Brinda algunas ventajas sobre el formato basado en archivos, como por ejemplo se puede acceder directamente a un mail y en las tareas de borrado solo se afecta el archivo que contiene el e-mail a borrar.
Aún así presenta inconvenientes, pues necesita de un mecanismo de bloqueo para asignar el nombre/número de los nuevos archivos que se generan al agregar nuevos e-mails.

Mx

Esta variante es similar a la anterior con la salvedad que se agregan como parte del formato ciertos archivos auxiliares, Estos archivos auxiliares facilitan la tarea de agregar nuevos e-mails.

Maildir

Es la variante más reciente de este tipo de formato; agrega algunas extensiones que lo hacen más robusto frente sus antecesores. Esta diseñada para resolver los problemas de file locking.

En lugar de poseer un solo directorio, *maildir* maneja tres directorios: ***new***, ***cur*** y ***tmp***. Cuando se agrega un nuevo e-mail primeramente pasa por el directorio ***tmp***, donde se le asigna un nombre único. Luego cuando el nuevo e-mail está completamente escrito es movido al directorio ***new***. Cuando los programas de lectura de e-mails (MRA) acceden a los correos almacenados en el ***new***, los e-mails se mueven al directorio ***cur*** junto a los otros mensajes ya leídos.

La gran ventaja de este tipo de formato es que cuando se quiere acceder a un e-mail en particular, la operación se puede realizar sin la necesidad de parsear todo un archivo para encontrarlo.
En las operaciones de borrado e inserción de nuevos e-mails no se afecta al resto de los e-mails del mailbox, solucionando el problema de que el mailbox pueda quedar corrupto por alguna causa.

Una desventaja, frente a los formatos basados en archivos, se puede dar en operaciones que tengan que leer o buscar información contenida en el header o body de los e-mails, pues en un formato basado en archivos se abre un solo archivo y en este formato tenemos que abrir y cerrar muchos archivos (uno por cada e-mail). Este problema puede resultar más notorio cuando el mailbox contiene una gran cantidad de e-mails o el soporte físico de los datos (disco duro) es lento.

Productos del mercado

Se presentan las principales características de productos que soportan los formatos antes descriptos.

Están agrupados por tipo de agente.

MTAs

Sendmail

Su versión actual es 8.13.4 y se puede encontrar en http://www.sendmail.org.
Es uno de los primeros MTAs en implementar el protocolo SMTP. Viene por defecto en numerosas instalaciones Unix.
Tiene la reputación de ser un sistema complejo de administrar pero completo. En sus últimas versiones se ofrecen macros y directivas de configuración, que hacen más sencilla las tareas de administración.
Su diseño es totalmente monolítico, aunque desde hace unos años ha sido mejorado.
Mediante la configuración se le puede indicar que LDA utilizar, permitiendo de esta manera una independencia entre el MTA y el formato de almacenamiento utilizado.
Se encuentra mucha documentación dedicada de este producto en internet.

Qmail

Su versión actual es 1.03 y se puede encontrar en http://www.qmail.org.
Es más reciente que el anterior. Fue diseñado para Unix, como un reemplazo de *sendmail* enfocando su desarrollo en la seguridad. No posee todas las funcionalidades que ofrece *sendmail*.
Su diseño es totalmente modular y se incluye en la distribución un LDA llamado qmail-local que soporta los formatos de almacenamiento *maildir* y *mbox*.
Se encuentra abundante documentación sobre su diseño e implementación

Postfix

Su versión actual es 2.2 y se puede encontrar en http://www.postfix.org.
Se propone como una alternativa rápida, fácil de configurar y segura para sustituir a *sendmail*. Posee menos funcionalidades que éste.
Se incluye en la distribución un LDA que soporta *mbox* y *maildir*.

Exim

Su versión actual es la 4.50 y se puede encontrar en http://www.exim.org
Es un descendiente de *sendmail*. A pesar de su diseño monolítico, ha centrado la atención en reforzar la seguridad siendo en este sentido comparable a *qmail* y *postfix*.
Incluye un LDA denominado appendfile que soporta los formatos *mbox*, *maildir* y *mbx*

MMDF

Su versión actual es 2.40 y se puede encontrar en http://mmdf.sourceforge.net
Este agente trabaja sobre el formato *mmdf*. (Se encuentra desactualizado)

RLAs

UW-Imap

Su versión actual es imap-2004c y se puede encontrar en http://www.washington.edu/imap
Posee un diseño modular. Su implementación se basa en una API llamada *C-Client*, dicha API utiliza un mecanismo de driver para acceder a mailboxes de diferentes formatos. Inicialmente existían drivers únicamente para los formatos *mbx* y *mbox*, con el correr del tiempo se han ido agregando drivers para soportar otros.

Courier-Imap

Su versión actual es 4.0.2 y se puede encontrar en http://www.courier-mta.org/imap/
Este agente brinda soporte específicamente para el formato *maildir*.
Es parte de un proyecto de mayor envergadura denominado *courier-server*.

Cyrus

Su versión actual es 2.2.12 y se puede encontrar en http://asg.web.cmu.edu/cyrus/imapd/.
Maneja su propio formato de almacenamiento y esta pensado para correr en servidores tipo caja negra.
Dado que maneja un formato propietario, brinda un LDA para que pueda ser utilizado por diferentes agentes MTA, por ejemplo *sendmail*.
Soporta varias extensiones del protocolo IMAP.

Dovecot

Su versión actual es 0.99.14 y se puede encontrar en http://www.dovecot.org.
Soporta el formato *mbox* y *maildir*. Tiene previsto brindar soporte para formatos basados en bases de datos.

Agentes mixtos (con almacenamiento en bases de datos)

Dbmail

Su versión actual es 2.1.0 y se puede encontrar en www.dbmail.org.
Este producto soporta los formatos basados en bases de datos, los DBMS que están actualmente soportados son MySQL y PostgreSQL. El DBMS a utilizar se fija en el momento de la compilación.
En la documentación no se encuentran análisis de desempeño ni comparaciones con otros formatos de almacenamiento.
En la distribución se incluye un LDA para que pueda ser utilizado con diferentes agentes MTA. También se implementa un agente MRA que soporta los protocolos IMAP y POP.

Oryx Mailstore

Su versión actual es 0.93 y se puede encontrar en http://www.oryx.com/mailstore/.
El producto está integrado por un conjunto de programas que proveen acceso a un formato de almacenamiento basado en bases de datos. Actualmente los e-mails son almacenados en una base de datos PostgeSQL, y en un futuro proveen incorporar soporte para otros manejadores.
Esta implementado utilizando el lenguaje de programación C++.

James

Su versión actual es 2.2.0 y se puede encontrar en http://james.apache.org.
Esta solución soporta archivos de texto y bases de datos.
Está escrito en el lenguaje Java.
Actualmente, solo soporta el protocolo POP3.

Utilitarios para el almacenamiento en bases de datos

Mail2Db

Es un LDA que almacena e-mails en una base de datos PostgreSQL.
Su versión actual es 1.15 y se puede encontrar en
ftp://ftp.tummy.com/pub/tummy/Mail2DB/
Este producto se encuentra desactualizado ya que no sufre cambios desde noviembre de 2001

Mbox2mysql

Es una herramienta que toma un mailbox en formato *mbox* y almacena los e-mails en una base de datos. Su versión actual es 0.3 y se puede encontrar en http://mbox2mysql.sourceforge.net.
Permite clasificar, buscar y filtrar mensajes mediante consultas SQL a la base.
También ofrece pasar al formato *mbox* los mensajes que están en la base de datos.

Trabajos relacionados al proyecto

En esta sección se comentan dos trabajos que se encuentran relacionados con los temas tratados en este proyecto y que resultaron de interés a los autores.

El primero de ellos es el llevado adelante por Nick Elprin y Bryan Parno *"An Análisis of Database-Driven Mail Server"*[42]. El trabajo fue presentado en el LISA (*Large Installation Systems Administration Conference*) en octubre del 2003.
En el se comparan tres servidores IMAP, soportando cada uno de ellos un mecanismo de almacenamiento diferente: *cyrus* trabajando sobre un formato propietario basado en BerkeleyDB[25], *uw-imap* con el formato *mbox*[15] y *courier-IMAP* utilizando *maildir*[16]. Además de estos servidores se utiliza el DBMS MySQL para simular el formato de almacenamiento basado en bases de datos. En este trabajo, por diferentes razones, no se integra el formato basado en bases de datos a ningún servidor, *"our time and resource constraints made it difficult to integrate mySQL storage into an existing IMAP Server"*[1].
En general, y por los resultados obtenidos en las pruebas, concluyen que sistemas de almacenamiento basados en base de datos son una buena solución comparados con los formatos tradicionales.
Un punto interesante, además del anterior, es que bajo sus condiciones de trabajo el formato *mbox* obtiene mejores resultados que el formato *maildir*.

El segundo de los trabajos expuestos es el llevado adelante por Sam Varshavchik *"Benchmarking mbox versus maildir"*[43], en este trabajo se presentan los resultados de una serie de pruebas comparativas entre los formatos *mbox* y *maildir*.
Varshavchik llega a la conclusión de que exceptuando casos específicos, *maildir* ofrece mejores resultados que *mbox.* Los casos en los que *maildir* se ve afectado son aquellos en los que el hardware sobre el cual está funcionado el servidor es lento o que el manejo del sistema de archivos es ineficiente. En estos casos *maildir* se encuentra afectado sobretodo en operaciones de búsquedas de campos específicos de los e-mails.

Si observamos los resultados de estos dos trabajos se puede ver que llegan a resultados opuestos en lo que respecta a *maildir* y *mbox*. Esto puede deberse a que justamente las condiciones de trabajo del primer enfoque coincide con uno de los casos que el segundo considera excepcional. Los mailbox de trabajo del primer enfoque contienen en el peor caso 21.000 e-mails, composición que seguramente afecta el desempeño del formato *maildir* en todo lo relacionado con búsquedas y accesos a partes específicas de e-mails.

Se agrega como comentario final a esta sección, que en el proyecto se realizaron diferentes pruebas de desempeño sobre varios formatos, los resultados de estas pruebas se exponen en el capítulo V del presente documento.

[1] En el presente proyecto, el formato basado en bases de datos es integrado a servidores de correo existentes. Más información se puede encontrar en el capítulo IV de este documento.

Capítulo III

Análisis del problema y solución propuesta

Resumen

En la primera sección de este capítulo se introducen los conceptos englobados en la realidad del problema, para luego proponer una solución. Más detalles se pueden encontrar en el: "Apartado I – Documento de Análisis".

Análisis de la realidad

Como parte del análisis comenzaremos definiendo los conceptos que manejaremos, ellos son: E-mail, Folder y Depósito de e-mail (Storage).

E-mail

Se entiende por este concepto a un mensaje que se transmite en forma electrónica entre usuarios.
Está compuesto por un cabezal (header) y un cuerpo (body).
El header está compuesto por un conjunto de campos (header fields), y cada campo tiene un nombre y un valor, más información sobre los headers se puede encontrar en el RFC 2076[27].
El body contiene la información que se transmite en el e-mail, en un principio estaba compuesta únicamente de texto plano (sin formato), hoy en día, debido al avance de la tecnología se ha hecho posible transmitir otro tipo de información, como lo pueden ser archivos de video, sonido, etc., más información sobre este punto se puede encontrar en la definición de MIME, en los siguientes RFCs: RFC 2045[28], RFC 2046[29], RFC 2047[30].
Para ahondar sobre el formato de un e-mail se puede consultar el RFC 822[31].

Folder

Un folder agrupa e-mails, pertenece a un usuario y tiene un nombre, también nos referiremos a los folder mediante el nombre de Mailbox.
Inbox es el nombre que se le da al folder donde van a parar los mensajes nuevos del usuario.
Dependiendo del formato de almacenamiento, un folder puede contener subfolders.

Depósito de e-mails (Storage)

Como se mencionó en el capítulo II el storage es el soporte "físico". En él es donde se localizan los folders. Puede ser un filesystem, una base de datos, etc.

Recordemos que los agentes que actúan directamente sobre los e-mails almacenados son el LDA y el MRA; el primero de ellos teniendo como tarea básica agregar nuevos e-mails y el segundo realizar consultas sobre la estructura de folders del usuario.

Modelo conceptual de la realidad

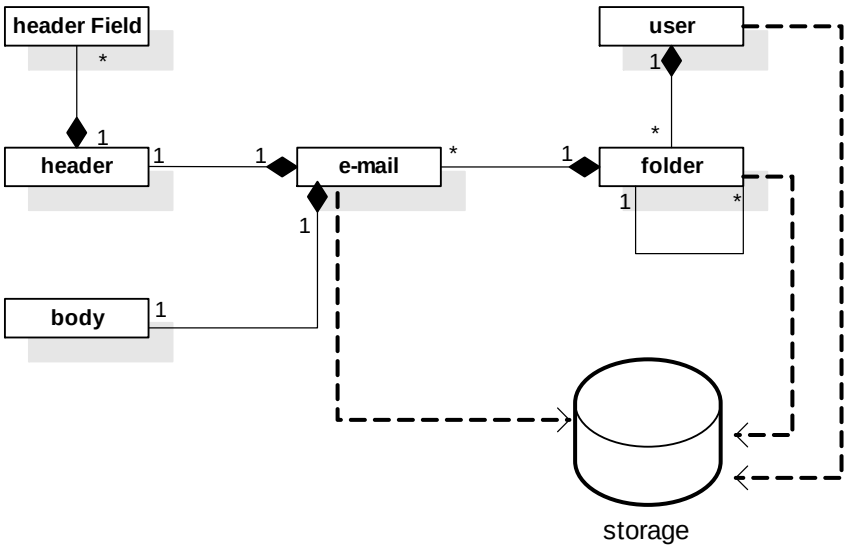


Figura III-1 Modelo conceptual de la realidad

El Problema

Planteada la realidad, el problema consiste en desarrollar un sistema que brinde soporte de almacenamiento a la estructura de folders del usuario, utilizando como storage una base de datos.
El sistema debe brindar las operaciones necesarias para que los agentes LDA y MRA lleven adelante sus tareas.

Requerimientos funcionales

Debido a que no existe información alguna sobre los requerimientos de un sistema de estas características y que tampoco se contaba con usuarios a los cuales consultar[2], como en un análisis de requerimientos estándar. Los requerimientos funcionales se obtuvieron del estudio de los agentes que interactúan directamente son el storage, el agente MRA y el LDA.

El requerimiento principal que se obtiene del agente LDA es el de agregar nuevos e-mails en determinado folder de un usuario.

Como mencionamos anteriormente, el MRA es un intermediario entre el MUA del usuario y el storage. Para comunicarse, el MRA y el MUA utilizan los protocolos IMAP y POP.
La aplicación desarrollada en este proyecto brinda los mecanismos necesarios para que el MRA obtenga los datos del storage, estos datos son utilizados para responder las consultas que realiza el MUA.

[2] Esto se debe a que los usuarios de este sistema son a su vez otros sistemas.

Consecuentemente, para identificar los requerimientos necesarios de la aplicación, se estudiaron los protocolos IMAP y POP.
Luego del estudio de estos protocolos, se concluyó que los requerimientos que se desprenden del protocolo POP son un subconjunto de los que se desprenden del protocolo IMAP.

Aclarado esto último, pasamos a describir básicamente los conceptos de una sesión de un MRA que se comunica mediante el protocolo IMAP, de este estudio es que salen la mayoría de los requerimientos funcionales que le solicitaremos a nuestro sistema.

A los MRAs que se comunican mediante el protocolo IMAP los llamaremos *servidores imap*.

En el transcurso de una sesión el servidor imap pasa por una serie de estados, estos estados son: *no authenticated, authenticated. y selected.*

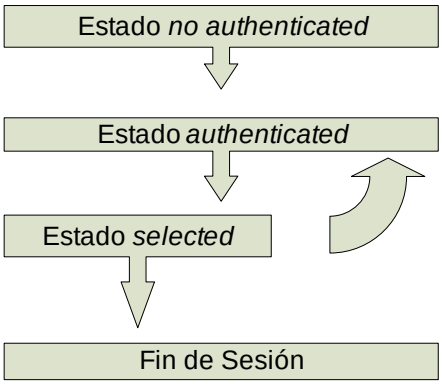


Figura III-2 Estados de una sesión IMAP

Una sesión no puede estar en más de un estado simultáneamente. En un estado se tiene acceso a un conjunto de operaciones, existen operaciones que sólo se pueden realizar en determinado estado, por más información sobre los estados de una sesión IMAP se puede referir a *"Anatomy of an IMAP Session"* [32]

Estado no authenticated
Es el estado inmediato, luego de que el MUA establece una conexión con el servidor imap.

Las operaciones o comandos que se pueden ejecutar en este estado son:

> **Authenticate** *mechanism*
> Este comando autentica un usuario ante el servidor imap, utilizando el mecanismo de autenticación especificado en el parámetro.
>
> **Login** *username password*
> Este comando autentica un usuario ante el servidor imap, utilizando la passaword en texto plano (sin encriptación).

Estado autenticated
Si estando en el estado *no authenticated* ejecutamos alguno de los comandos de autenticación y obtenemos un resultado satisfactorio, pasamos al estado authenticated. Luego de que pasamos a este nuevo estado no se puede retornar al estado anterior.

Los comandos validos en este estado son:

Append *mailbox-name*
Agrega un nuevo e-mail al mailbox cuyo nombre es especificado como parámetro

Create *mailbox-name*
Crea un nuevo mailbox con el nombre especificado como parámetro

Delete *mailbox-name*
Borra en forma permanente el mailbox cuyo nombre es especificado como parámetro.

Rename *mailbox-name newmailbox*
Cambia el nombre del mailbox-name a newmailbox

List *basename mailbox*
Retorna una lista de nombres de mailboxes que coinciden con el nombre pasado como parámetro. El parámetro puede contener caracteres especiales como lo son "*", etc. *Basename* refiere al espacio de nombres en el cual estamos buscando.

Lsub *basename mailbox*
Similar al comando anterior con la excepción de que opera sobre los mailbox en los cuales el usuario esta subscripto.

Subscribe *mailbox-name* y **Unsubscribe** *mailbox*
El primero de ellos agrega el mailbox cuyo nombre es pasado como parámetro a la lista de mailboxes subscriptos por el usuario. El segundo realiza la operación inversa.

Select *mailbox-name* y **Examine** *mailbox-name*
Seleccionan el mailbox de trabajo, los e-mails con los cuales se va a trabajar son los que pertenecen al mailbox seleccionado. La diferencia entre *select* y *examine* radica en que en el primero de estos comando se accede a los e-mails en modo lectura/escritura y en el segundo en modo lectura.

Estado selected
Este estado comienza cuando estando en el estado *authenticated* se ejecuta el comando select o examine en forma exitosa. Se puede retornar al estado anterior ejecutando el comando close.

Los comandos validos en este estado son:

Close
Cierra el mailbox en el cual se está trabajando, borrando todos los e-mails que están marcados para borrar. Se retorna al estado *authenticated*.

Copy *e-mails mailbox-name*
Copia los e-mails, del mailbox con el que se está trabajando, al mailbox cuyo nombre es pasado como parámetro. El parámetro *e-mails* refiere a una lista numérica, donde los números significan la posición relativa de e-mail con respecto al inicio, por ejemplo: **1** significa el primer e-mail, **1:*** significa todos los e-mails del mailbox.

Fetch *e-mails datos*
Retorna los datos que le son solicitados como parámetro, de los e-mails del mailbox en el cual se está trabajando. El parámetro *e-mails* es ídem al anterior.

Search *criterio*
Retorna la lista de los números de mensajes (del mailbox en el cual se está trabajando) que coinciden con el criterio de búsqueda.

Store *e-mails data*
Cambia las banderas, de los *e-mails* especificados, a los valores que se especifican en data. En data se especifica el nombre de la bandera a cambiar y el valor que se le quiere asignar, por ejemplo *store 1 +flags (\deleted)* pone la bandera *deleted* activa para el primer e-mail.

Expunge
Borra permanentemente del mailbox los e-mails que tienen la bandera *deleted* activa.

Las operaciones anteriores no son las únicas soportadas por el protocolo IMAP, para ver el resto o para obtener una descripción mas detallada de las expuestas, dirigirse al RFC 2060 [19]

Los requerimientos funcionales que se desprenden del estudio, se pueden agrupar en las siguientes categorías:

- **Requerimientos sobre las Sesiones**
  R 1. Proporcionar algún mecanismo de autenticación.

- **Requerimientos sobre los Mailboxes**
  R 2. Obtener la lista de mailbox de un usuario.
  R 3. Poder crear, borrar y renombrar mailbox.
  R 4. Seleccionar el mailbox de trabajo.
  R 5. Agregar nuevos e-mails a un mailbox.
  R 6. Obtener un e-mail de un mailbox.
  R 7. Buscar dentro de un mailbox los e-mails que cumplen con determinado patrón.
  R 8. Borrar de un mailbox todos los e-mails que estén marcados para borrar.

- **Requerimientos sobre los Emails**
  R 9. Cambiar las banderas de estado de un e-mail.
  R 10. Obtener las diferentes partes de un e-mail (banderas, headers, y partes del body).

- **Requerimientos sobre los Usuarios.**
  R 11. Crear nuevos usuarios.
  R 12. Cambiar password de un usuario.
  R 13. Borrar Usuarios.

Requerimientos no funcionales

Rnf 1. Trabajar con una base de datos como storage.

Rnf 2. Ofrecer flexibilidad en la elección del DBMS que se utilizará como storage. Utilizar el DBMS deseado debe implicar un mínimo de programación adicional

Rnf 3. Ofrecer servicios a los agentes LDA y MRA sin depender de la implementación particular de éstos. Además debe requerir un mínimo de programación adicional par poder utilizar la implementación deseada de cada uno de estos agentes.

Rnf 4. Ofrecer independencia del sistema operativo sobre el cual se está trabajando.

Rnf 5. Incluir una interfaz de administración que actúe sobre los datos que almacena el sistema.

Actores y Casos de Uso

Actores

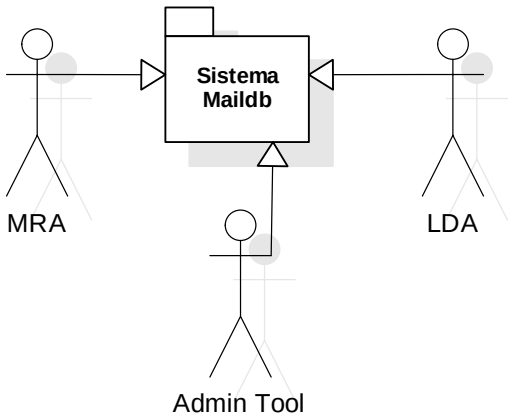


Figura III-3 Actores

MRA y el LDA: Son los actores primarios, es decir son los que están en contacto permanente con el sistema.

El Admin. Tool: Es un actor secundario del sistema, es la herramienta que se encarga de las tareas de mantenimiento del mismo, como por ejemplo: la creación de nuevos usuarios.

Casos de Uso

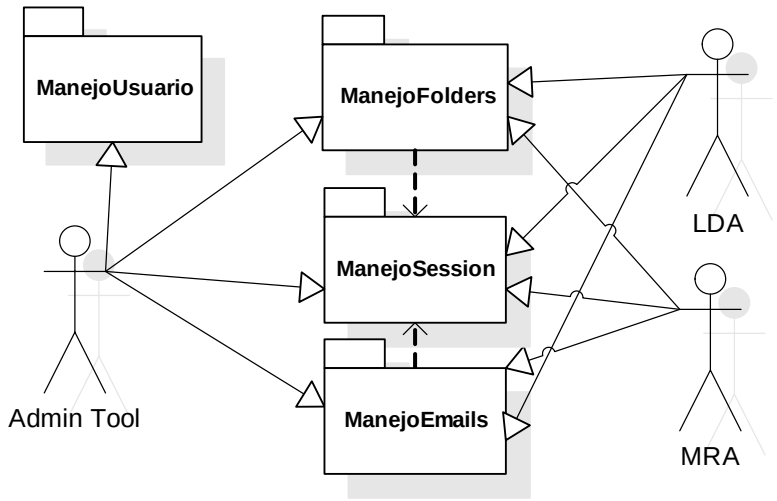


Figura III-4 Casos de Uso

Este diagrama es una representación en alto nivel, en él están representados los grandes grupos de casos de uso que se identificaron en el análisis del sistema, por más detalles sobre los casos de uso específicos dirigirse al *"Apartado I – Documento de Análisis"*

Arquitectura y diseño

A continuación se presenta un resumen del diseño de la aplicación. Los detalles completos se pueden encontrar en el "Apartado II – Documento de Diseño".

La Maildb API (Application Program Interfaz) proporciona a los desarrolladores de LDAs y MRAs los medios necesarios para solicitar servicios y comunicarse con el almacenamiento de e-mails, cuyo formato es una base de datos.
Esta API está compuesta por dos módulos o librerías, el módulo de presentación y el de persistencia.

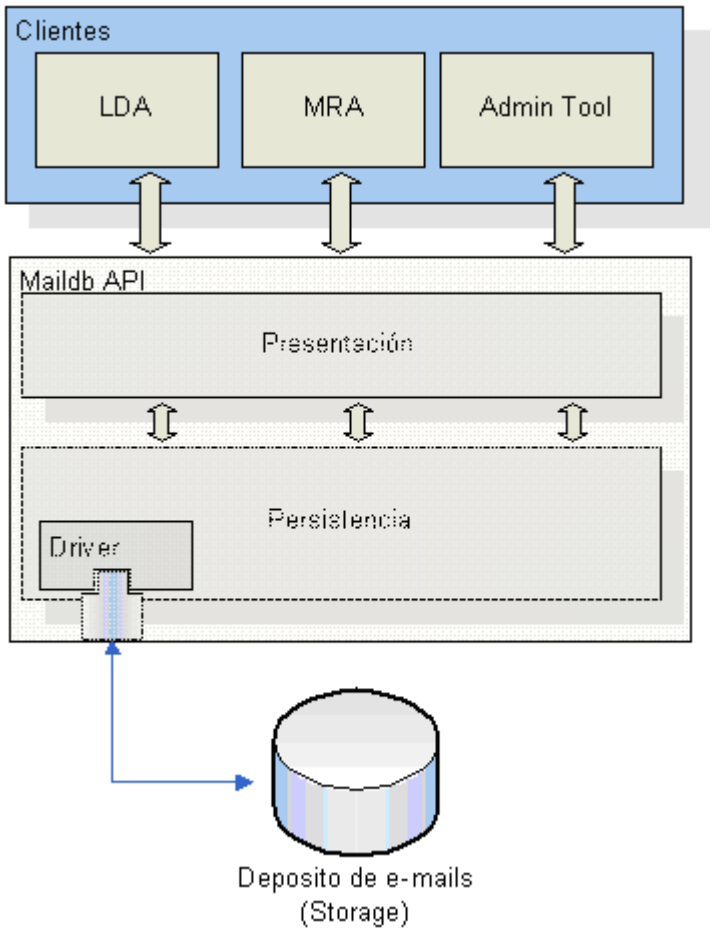


Figura III-5 Arquitectura general del sistema

Módulo de persistencia

Comenzamos describiendo el módulo de persistencia ya que es el núcleo (core) del sistema, es el encargado de almacenar y obtener los datos que se manejan.

Para trabajar con el almacenamiento de datos es necesario definir y determinar los componentes que integran este módulo.

Se pueden dividir en tres categorías:

1. El storage.
2. Componente de consulta.
3. El tipo de datos retornados.

El *storage* es un componente que está predefinido por el proyecto, y se trata de una base de datos, la idea fundamental del trabajo llevado adelante es utilizar este tipo de almacenamiento y compararlo con otros.
El requerimiento más importante solicitado sobre el storage, también uno de los más importantes de todo el sistema, es que si bien se trabaja con una base de datos, se pide no estar ligado a ningún manejador específico, como lo puede ser Oracle[33], MySQL[34], PostgreSQL [36] o algún otro. Al introducirnos en el detalle de diseño, en secciones siguientes de este capítulo, daremos a conocer el enfoque seguido para cumplir con este requerimiento.

El *componente de consulta* es el encargado de brindar los métodos necesarios para consultar y almacenar los datos de la aplicación, encapsulando la lógica y la tecnología utilizada para tal fin. Los métodos brindados, por las clases que integran este componente, son los que comúnmente utilizamos al acceder a una fuente de datos, crear, leer, borrar y actualizar.

Luego de realizar una consulta, por intermedio del *componente de consulta*, se tendrán que comunicar los resultados, el tipo de datos retornado también es un punto importante a definir y detallaremos su diseño más adelante.

Para ilustrar de manera gráfica los conceptos manejados en este punto se puede observar la siguiente figura.

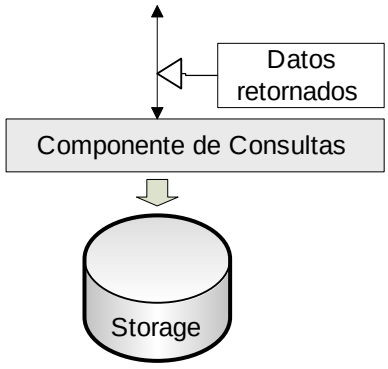


Figura III-6 Módulo de persistencia

Storage

Comenzaremos a detallar el diseño del storage y los mecanismos para independizarnos de cualquier tecnología existente en lo que refiere a Data Base Management System (*DBMS).*

Como comentamos anteriormente el tipo de storage utilizado es una base de datos.

Los DBMS proporcionan muchas facilidades de administración, capacidades de seguridad y operaciones sobre los datos que ellos manejan, además de lenguajes de consultas como por ejemplo SQL. El diseño y estudio detallado de DBMS esta más allá del alcance de este proyecto y nos remitiremos solamente a utilizarlos y aprovechar la funcionalidad que estos nos proporcionan.

Habiendo aclarado este punto y siguiendo con el estudio del diseño, se incluye un nuevo componente al módulo de persistencia que sirve de ayuda al momento de encapsular el DBMS utilizado.

Este nuevo componente actúa como un *bridge* o puente entre el *componente de consulta* y el DBMS. Es un componente que se puede intercambiar fácilmente sin necesidad de reestructurar toda la aplicación en cada oportunidad, proporcionando de esta manera un mayor nivel de integración con el ambiente que utilice la aplicación.

Para cumplir con este requerimiento, se maneja un mecanismo conocido como *driver o manejador*.

En el diseño se incluye una interfaz que define las operaciones necesarias para interactuar con los DBMS, luego para cada DBMS distinto se implementa esta interfaz y cada una de estas implementaciones se llama *driver del DBMS.*

En la siguiente figura se observa la arquitectura que surge de este diseño

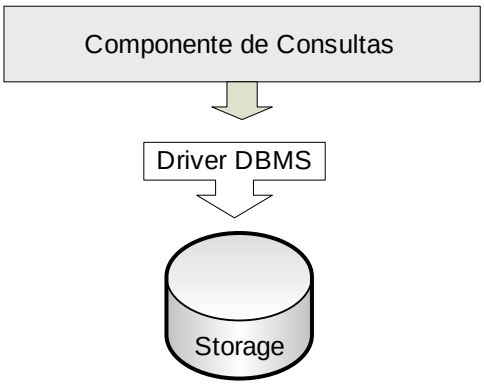


Figura III-7 Arquitectura de Drivers DBMS

Para darle más potencia a la aplicación y proporcionar un nivel de independencia aún mayor se exige otro requerimiento, no necesitar recompilar la aplicación cada vez que se quiera cambiar de driver de DBMS, es decir, se necesita que los drivers sean acoplados a la aplicación en tiempo de ejecución.

Esto conduce a diseñar un conjunto de componentes para brindar el mecanismo de carga dinámica de drivers.

Mecanismo de carga dinámica.

En el diseño de este mecanismo se utiliza el patrón de diseño Factory [37], este patrón se encarga de brindar componentes a quien se lo solicite ocultando la forma en que éstos son obtenidos.
Cada uno de los drivers se implementa como una librería que se enlaza con la aplicación en forma dinámica.
El Factory DBMS (DbConnectionFactory) se encarga entonces de localizar la librería dinámica y enlazarla con la aplicación poniendo a disposición, en tiempo de ejecución, los componentes que en ella se implementan, en este caso el driver del DBMS (DbConnection) a utilizar.

Componente de consulta.

Este componente surge con el fin de proporcionar un acceso simple a los datos manejados por la aplicación, brindando las operaciones necesarias para acceder a dichos datos.
Oculta el formato de los datos al resto de la aplicación, toda la lógica de acceso a los datos esta encapsulada y centralizada en este componente, de esta manera se facilita el mantenimiento del sistema. En otras palabras, este componente es el único que tiene conocimiento de la estructura utilizada para el almacenamiento de los datos.
En nuestro caso conoce las estructuras de las tablas de la base de datos y cuales son las relaciones entre ellas, conoce también el lenguaje de consulta que se utiliza.
No accede a los datos almacenados en el storage directamente, lleva adelante su tarea por intermedio de los drivers.

Las tres clases más destacadas de este componente son:

FolderQuery.
Se encarga de encapsular las operaciones relacionadas con el acceso a los datos de los folders.

MessageQuery.
Se encarga de encapsular las operaciones relacionadas con el acceso a los datos de los e-mails.

UserQuery.
Se encarga de encapsular las operaciones relacionadas con el acceso a los datos de los usuarios.

Datos retornados por el componente de consulta

El componente de consulta necesita comunicar sus resultados a quienes lo utilizan.
Por esta razón, se diseño un componente que se encarga del transporte de los datos a través del sistema, este componente se denomina DATA.
DATA funciona como una unidad independiente del storage utilizado, es decir, un objeto de tipo DATA se lo puede considerar como un conjunto de registros que siempre está desconectado y que no sabe nada sobre el origen y el destino de los datos que contiene.
Este tipo de datos no solamente se utiliza para comunicar los resultados del componente de consulta con el exterior, también se utiliza en la comunicación entre este último y el driver que se encarga de encapsular el storage.
El driver retorna, luego de una operación, un objeto de tipo DATA que es "llenado" por él.

Un diagrama de clases resumido de este módulo es el siguiente.

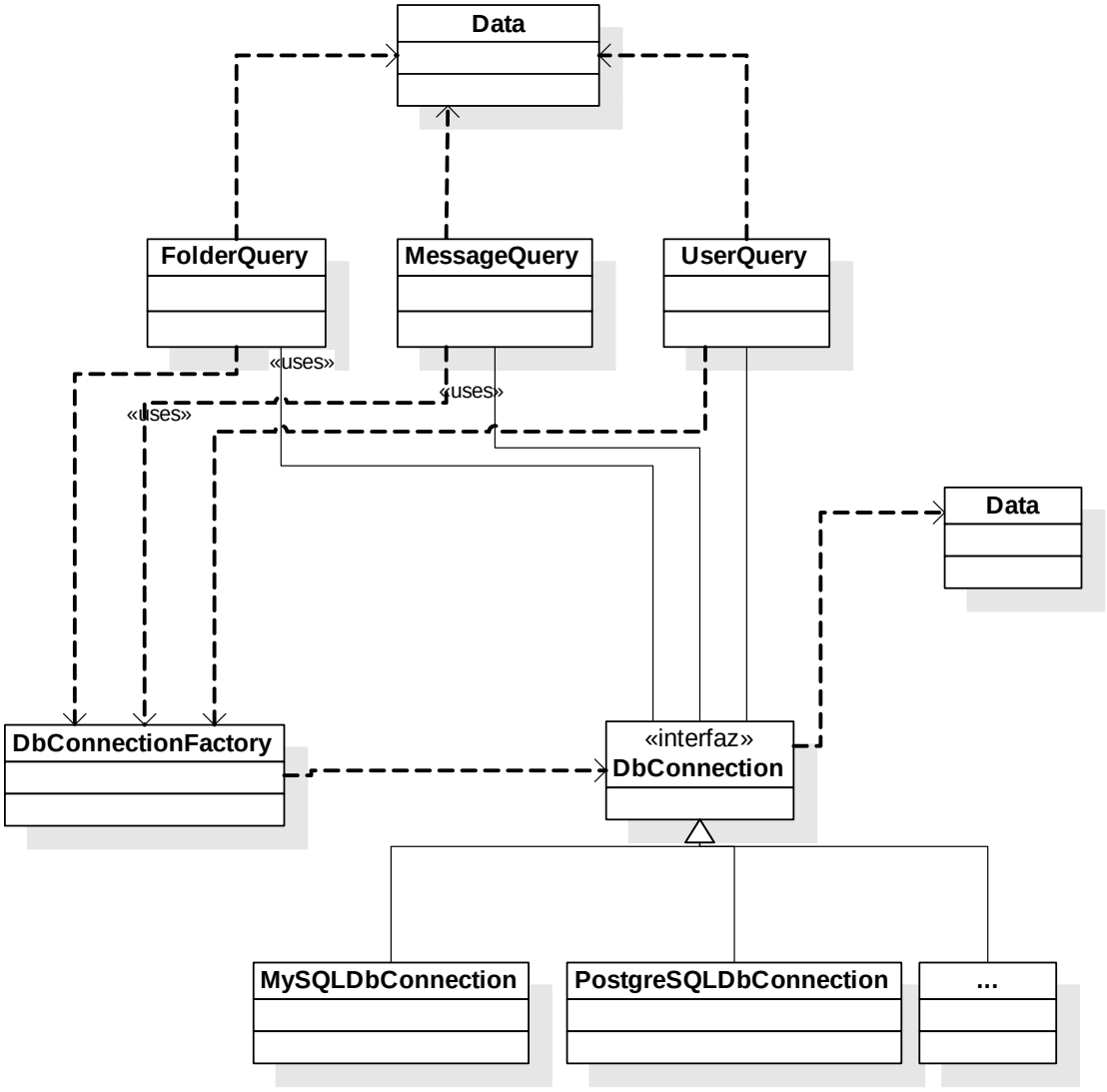


Figura III-8 Diagrama de clases del módulo de persistencia

Módulo de presentación

Este módulo brinda un nivel más de abstracción de los datos y nos aproxima al módelo de realidad, se encarga de mantener las relaciones de los datos y brindar una interfaz más amigable para el programador que la utilice.

Está integrado por un conjunto de clases dentro de las cuales podemos destacar las siguientes: session, folder, message y connection.

En la siguiente figura se presenta un diagrama resumido de las clases que componen este módulo.

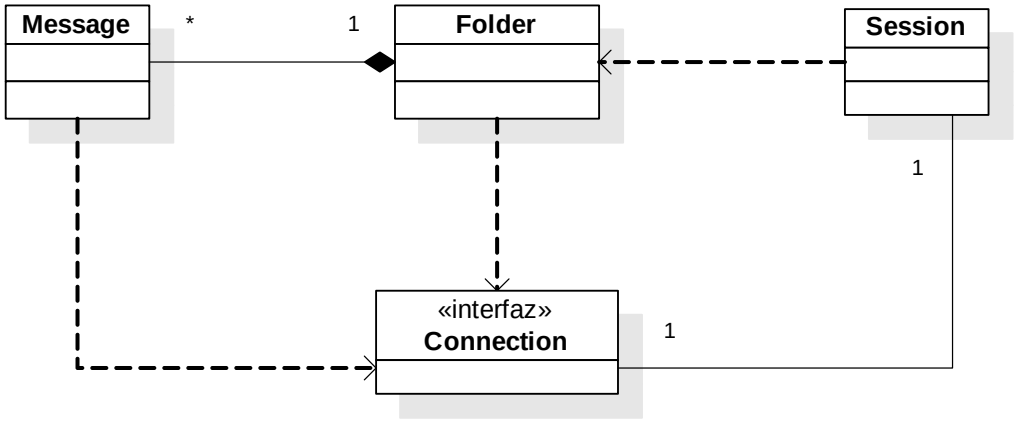


Figura III-9 Diagrama de clases del módulo de presentación

Session

Esta clase es el punto de entrada al modulo, es la encargada de establecer la conexión con el módulo de persistencia y mantener la información de autenticación del usuario.

Folder

La clase folder define un conjunto de atributos y métodos para acceder a los folder de un usuario.

Message

La clase message define un conjunto de atributos y métodos para acceder a los e-mails.

Connection

Es una interfaz que encapsula el mecanismo de conexión entre el módulo de presentación y el módulo de persistencia, actúa como un gateway entre éstos. De esta manera, las demás clases que componen el presente módulo no interactúan directamente con el módulo de persistencia.

La finalidad buscada al introducir esta interfaz es el de abstraer a las demás clases que integran este módulo del camino que se recorre para llegar a la fuente de datos, centralizando la comunicación en un único punto. Se logra además que los módulos de persistencia y presentación queden débilmente acoplados, permitiendo que evolucionen

independientemente. Este diseño permite organizar el sistema de diferentes maneras, a continuación presentamos dos posibles ejemplos de arquitecturas.

Arquitectura cliente-servidor.

En esta arquitectura el servidor sería el encargado de conocer y consultar la fuente de datos para luego responderle al cliente. Para implementar el servidor se haría uso del módulo de persistencia y se tendría la necesidad de definir un protocolo de comunicación entre el servidor y los clientes.
Para implementar el cliente se utilizaría el módulo de presentación. *Connection* sería el encargado, por parte del cliente, de conocer el protocolo de comunicación con el servidor. Las demás clases del módulo de presentación no se verían afectadas.

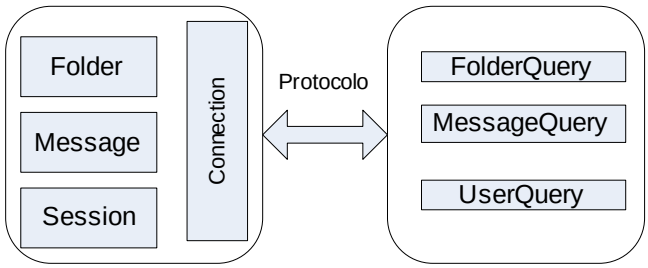


Figura III-10 *Connection* en una arquitectura cliente-servidor

Dentro de las ventajas de este enfoque podemos destacar que se pueden modificar los componentes del servidor en tiempo de ejecución, sin tener que afectar a los clientes. Se podría manejar en el servidor un pool de conexiones con la base de datos y el programador que utilice el sistema no tendría que preocuparse de estos detalles.

Conecction como Fachada del módulo de persistencia.

Se podría implementar *connection* (hablando en términos de patrones de diseño) como una facha del módulo de persistencia. En este caso, el sistema se vería reducido una librería que ofrecería una API de acceso a los datos.

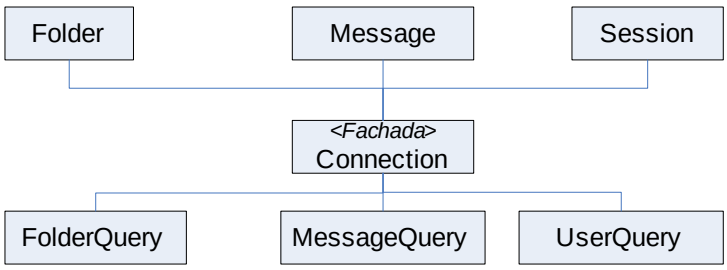


Figura III-11 *Conecction* como fachada del módulo de persistencia

Dentro de las ventajas de este enfoque se puede destacar que el sistema no es tan complejo y que no es necesario definir ningún protocolo de comunicación.

Capítulo IV

Desarrollo de la API e integración con el ambiente

Resumen

El sistema de almacenamiento de e-mails no es un ente aislado, está sumergido en una realidad en la cual existen otros actores involucrados. Dentro de estos actores se encuentran los agentes[3], éstos se encargan de brindar los servicios ofrecidos por un servidor de e-mails. Entonces, cuando se piensa poner en funcionamiento un prototipo del sistema encargado del almacenamiento de e-mails, es necesario también crear el ambiente que lo rodea.
En este capítulo se enumeran los diferentes componentes del sistema de e-mails y se justifican los motivos por los cuales fueron seleccionados.
Además, se describe las decisiones tomadas en cuanto a la implementación del prototipo del sistema diseñado en el proyecto, que llamamos Maildb API.

Contexto

Los componentes que integran el ambiente en el cual funciona el prototipo del sistema de almacenamiento de e-mails son los siguientes:

- Servidor de correo entrante (agente MTA)
- Agente LDA
- Servidor de correo saliente (agente MRA)
- Storage
- Maildb API (prototipo del sistema).

En la Figura IV-1 se observa un diagrama en el cual se identifican cada uno de los componentes del prototipo. A continuación en este capítulo se los pasarán a describir.

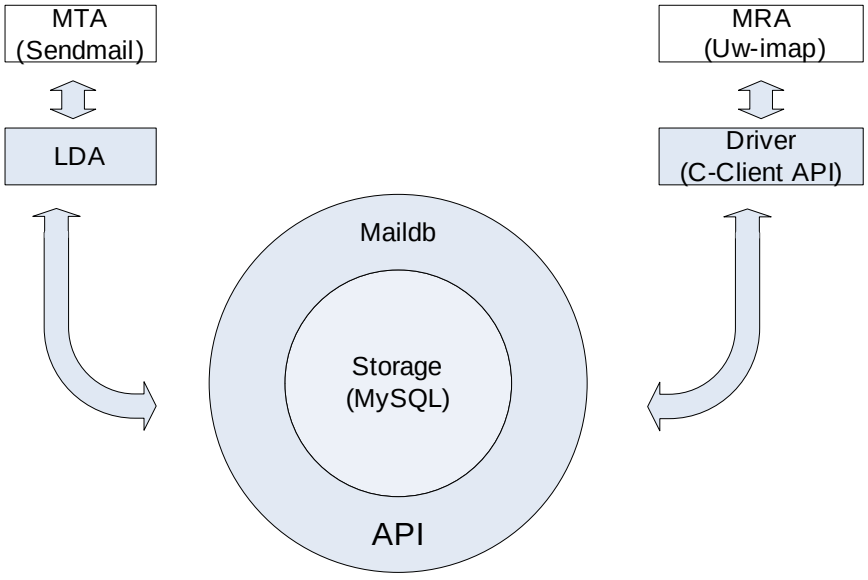


Figura IV-1 Prototipo

[3] Una descripción de los agentes se encuentra en el capítulo 2, sección Fundamentos del sistema de e-mails.

Agente MTA

Se eligió a *sendmail* como agente MTA del prototipo. La razón de su elección radicó en lo siguiente:

- Es el servidor de correo entrante utilizado en Facultad de Ingeniería y por ende, se cuenta con la experiencia del grupo de administración.
- Se le puede indicar, mediante su configuración, cual es el LDA a utilizar. Este último punto nos otorga más libertad al momento de elegir el LDA propiamente dicho.

Agente LDA

Dada la libertad que nos brinda el MTA, no se genera inconveniente alguno al momento de elegir el LDA.

La decisión pasa entonces entre elegir un LDA existente en el mercado y agregarle la funcionalidad de poder utilizar el formato de almacenamiento propuesto, o implementar uno propio.

Para poder tomar esta decisión se estudiaron requerimientos y funcionalidades de un LDA. Lo que comúnmente hacen los LDA es leer un texto de la entrada estándar junto con el usuario que es pasado como parámetro, luego controla que dicho texto se corresponda con un mail en formato RFC822. Por último, se anexa este nuevo e-mail al mailbox del usuario que le es pasado como parámetro.
El procedimiento utilizado para anexar el nuevo e-mail al mailbox del usuario depende del formato de almacenamiento de e-mail utilizado, por ejemplo, *procmail* soporta los formatos *maildir* y *mbox*.

En la búsqueda de información sobre implementaciones de LDAs no se encontró buena documentación. Entonces, si se quiere reutilizar algún LDA existente en el mercado se tiene que estudiar su código fuente. Consecuentemente, la opción de dotar a un LDA (existente en el mercado) con soporte para nuestro formato, no es una tarea sencilla e insume un tiempo que no se justifica.

Descartada la opción anterior, se decide implementar un LDA propio. Este LDA fue desarrollado en C++ y hace uso de la API diseñada en este proyecto. La opción de crear un LDA propio no pierde generalidad frente a la alternativa de reutilizar alguno.

Agente MRA

Implementar un MRA no es una tarea tan sencilla como la de implementar un LDA. Entonces la opción que queda es reutilizar alguno de los productos existentes en el mercado. La elección recayó sobre "*IMAP4rev1/C-Client Development enviroment*"[15], como su nombre lo indica nos estamos refiriendo a un ambiente de desarrollo, no sólo a un MRA.

Habitualmente se utiliza el nombre UW-IMAP para referir al servidor imap (MRA) que viene integrado a este ambiente de desarrollo. Junto a este servidor imap, se incluye un servidor pop, utilidades de testeo del protocolo IMAP y una API llamada C-Client [39]
El principal responsable de la implementación y el diseño de este ambiente de desarrollo es Mark Crispin.
Crispin comenzó a trabajar en la C-Client API en el año 1988 cuando realizaba tareas en la Universidad de Stanford. La primera versión del servidor IMAP basada en la C-Client API fue escrita por Crispin en 1990, trabajando en la Universidad de Washington, de ahí su nombre UW-IMAP (Universidad de Washington IMAP).

Crispin es un referente importante dentro del campo de la investigación de IMAP, Es responsable de escribir el RFC 2060 *Internet Message Access Protocol Versión 4rev1*, considerado uno de los documentos más importante en la definición del protocolo.

Dentro de las razones que llevaron a elegir este producto (el ambiente de desarrollo) se encuentran las siguientes:

- UW-IMAP es el servidor IMAP utilizado en Facultad de Ingeniería, esto facilita el proceso de instalación y configuración, pues se cuenta con la experiencia acumulada por el grupo de Administración de servidores.

- Los servidores que lo integran (IMAP y POP) están implementados en base a un diseño modular. Hacen uso de la C-Client API que tienen la particularidad de implementar un sistema de drivers mediante el cual accede a mailboxes de diferentes formatos, lo cual permite dotar al servidor IMAP (sin tener que modificarlo) con la capacidad de poder acceder al formato de almacenamiento propuesto en el proyecto. La tarea de integración del MRA con la API desarrollada en el proyecto se reduce a crear un nuevo driver.

- Existe documentación variada sobre el producto en Internet y en particular en el sitio de la Universidad de Washington[15]. Dentro de esta documentación se destaca la documentación técnica sobre la creación de nuevos drivers para la C-Client API

Storage

En el prototipo desarrollado se utilizó MySQL como DBMS para el soporte de almacenamiento. Fue elegido por la experiencia que se tenía en la instalación del mismo y además porque fue utilizado para realizar las pruebas de un producto de similares características que el implementado en el proyecto (Dbmail).
Se dispone también de buena documentación de MySQL en Internet tanto para el usuario como para el desarrollador de aplicaciones.

Maildb API

Lenguaje y herramientas de programación.

Nustro diseño está orientado a objetos, para implementar la API se utilizó el lenguaje C++. Este lenguaje fue seleccionado sobre otras alternativas (por ejemplo JAVA), principalmente por razones de rendimiento, además brinda implícitamente una capa de abstracción sobre el sistema operativo.

El desarrollo se llevó adelante en una plataforma Linux utilizando la herramienta llamada Kdevelop [38]

Decisiones de implementación

Para desarrollar un prototipo nos basamos en una metodología incremental. Se tomó la decisión de categorizar las operaciones, para luego desarrollar un núcleo que sirva de base en el desarrollo y paulatinamente agregarle funcionalidad. El criterio de categorización está basado en qué tan importante es cada operación para el funcionamiento de los agentes MRA y LDA.

Tenemos entonces los siguientes grupos de operaciones[4]:

Grupo 1

El grupo está integrado por aquellas operaciones que son imprescindibles para el funcionamiento de los agentes y por ende, para poder medir el desempeño del formato de almacenamiento.

- Login
- Logout
- Insert
- Select
- Close
- Fetch
- Store
- Expunge

Grupo 2

Aquí agrupamos operaciones que no son imprescindibles para el funcionamiento de los agentes. La operación *search* no es imprescindible, pero es deseable debido a que el desempeño de la misma podría verse beneficiada mediante el uso de un formato basado en bases se datos.

- Search
- Append

[4] Recordar que las operaciones IMAP fueron descriptas en el capítulo 2

- Create
- Delete
- Rename
- Copy
- List
- Lsub
- Subscribe
- Unsubscribe

Decidimos entonces implementar las clases y métodos que brinden soporte en principio a las operaciones del grupo 1 y que paulatinamente se pueda incorporar el soporte para el resto de las operaciones (grupo 2).

Lamentablemente, por razones de tiempo, solamente fue posible brindar la funcionalidad para poder dar soporte a las operaciones del primer grupo.

Módulo de presentación

En la etapa de prototipado de la aplicación se recopiló información de diferentes herramientas. Entre éstas se encontraron frameworks de trabajo orientados al área en la que estamos trabajando, algunas de ellos son *vmime* [50], *gmime* [51] y *libetpan* [52]. Estas herramientas fueron evaluadas en su momento y se decidió no hacer uso de ellas.
Se consideró que el tiempo insumido en aprender a utilizar dichas herramientas sería mejor aprovechado en otras tareas. Viéndolo en retrospectiva y con mayor experiencia en el tema, podría haber sido una buena opción la utilización de alguno de estos frameworks.

Una de las decisiones a tomar cuando estamos trabajando sobre el módulo de persistencia es cómo se implementa la interfaz *connection*[5].

Para el prototipo y hablando en términos de diseño orientado a patrones, la interfaz *connection* de este módulo fue implementada como una fachada [37] de las clases que integran el módulo de persistencia.

Módulo de persistencia

En lo que refiere a la implementación del módulo de persistencia, en el diseño se cuenta con clases específicas para el manejo del almacenamiento de cada uno de los elementos del sistema (folders, messages y users). En el prototipo se optó por desarrollar una sola clase que se encarga de brindar soporte a las consultas que describimos anteriormente como imprescindibles.

Sobre la plataforma que permite independizarnos del DBMS, se realizaron búsquedas de productos que cumplieran con este requerimiento o nos sirvieran de guía, sin obtener muchos resultados. La implementación de este mecanismo fue uno de los puntos acometidos primeramente, pues tenía relación con la carga de librerías dinámicas, técnica que sería utilizada en varios lugares dentro del desarrollo.

[5] Capítulo 3, sección *Módulo de presentación*.

Luego de implementado el mecanismo de drivers para DBMS, encontramos casualmente un producto que maneja este mismo concepto. Este producto se denomina *dbconnect*[48] y viene siendo desarrollado desde mediados del año 2000. Está dedicado pura y exclusivamente a mecanismos para independizar el desarrollo de aplicaciones de la base de datos utilizada, y se encuentra en una etapa madura. Cuenta con un diseño de características similares al encarado en el proyecto, con algunas diferencias.
La principal diferencia radica en la forma de retornar los resultados de una consulta. En el caso del proyecto, el driver retorna los resultados de las consultas en objetos de la clase DATA[6] (descrita anteriormente), siendo DATA totalmente independiente del driver utilizado, estableciéndose en este caso una relación de uso entre el driver y el tipo DATA.
En el caso de *dbconnect* se brinda una interfaz, similar a DATA, que el desarrollador tiene que implementar para cada driver. La principal ventaja de este enfoque es que esta interfaz es implementada en base a los tipos de datos proporcionados por los desarrolladores de DBMS.

Driver DBMS

Como comentamos anteriormente el DBMS utilizado para auspiciar de storage es MySQL. En el sitio de MySQL [34] existe un área dedicada a desarrolladores, en ésta se pueden encontrar APIs para diferentes lenguajes de programación.

Entre las APIs brindadas se encuentran la MySQL C API y la MySQL C++ API [35]. En principio, la utilización de una u otra no influye en el prototipo, pues una de los aspectos de Maildb API es el poder cambiar el driver DBMS sin afectar el resto de la aplicación.
Luego de estudiar las dos APIs brindadas por MySQL, para implementar el driver del prototipo se utilizó MySQL C API. La decisión esta motivada en que la MySQL C++API es en realidad un wrapper [35] de MySQL C API.

Modelo de datos

Se considera que este es uno de los puntos más interesantes del prototipo y del proyecto, pero debido a la diversidad de temas que se atacaron no se le pudo dedicar el tiempo suficiente. De todas maneras, dado que nuestro diseño es modular, los cambios que se realizan en la estructura de tablas de la base de datos, solo afectan al módulo de persistencia. Surge de aquí una buena propuesta para trabajo a futuro, que consiste en diseñar un módulo de persistencia, sea este un módulo de consultas inteligente o adaptativo. La inteligencia vendría dada por la existencia de un motor que "*esté al tanto"* de cambios en la estructura de la base de datos y que "*adapte"* sus consultas a la nueva estructura. El mecanismo de *"estar al tanto y adaptar sus consultas"* podría consistir sencillamente en que, en lugar de que las consultas SQL estén harcoded en cada operación éstas puedan ser tomadas, por ejemplo, de un archivo con estructura XML.

[6] Capítulo 3 , sección *Módulo de persistencia* (datos retornados por el componente de consultas)

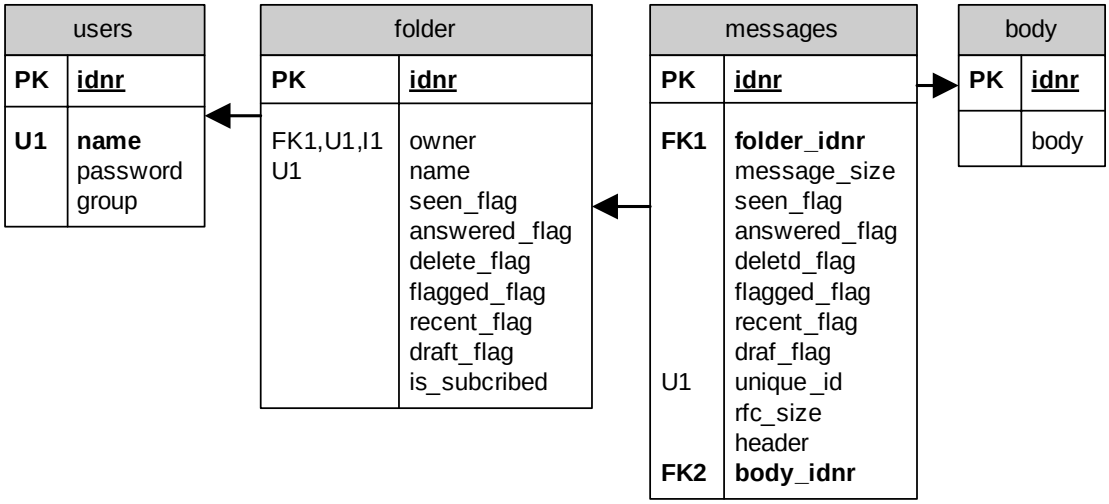


Figura IV-2 Modelo de datos del prototipo

En la Figura IV-2 se presenta la estructura de la base de datos utilizada en el prototipo. Está compuesta por cuatro tablas users, fólder, messages y body.

En la tabla users se tiene la información relacionada con el usuario, tiene un identificador único que es de uso interno, un nombre que también lo identifica, una contraseña y el grupo al que pertenece.

La tabla fólder esta compuesta por un identificador único de uso interno, un usuario dueño, un nombre y un conjunto de banderas (flags). El nombre del fólder es único dentro de los nombres de carpeta de un usuario.

Messages esta compuesta por un identificado único de uso interno, el identificador de fólder al cual pertenece, un conjunto banderas (flags), el identificador único de mensaje, el header y un identificador que apunta al body.

La tabla body se compone de un identificador único de uso interno y un campo de texto que es el body propiamente dicho.

Interconexión entre C-Client API y Maildb API.

Descripción general de la C-Client API

Esta API brinda a los desarrolladores un framework para implementar acceso a los mailboxes por intermedio de drivers. El desarrollador cuenta con un conjunto de estructuras y funciones[39] que hacen posible llevar adelante la implementación de nuevos drivers. Este framework esta implementado utilizando el lenguaje C.

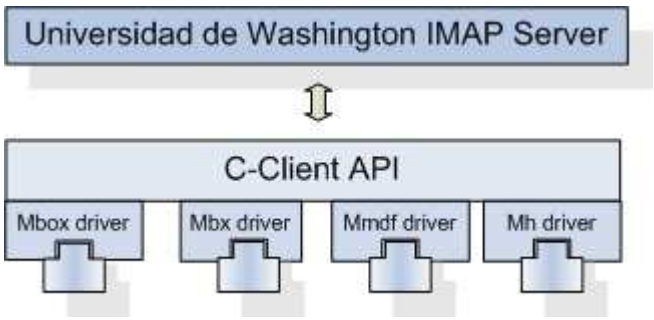


Figura IV-3 C-Client API

Wrapper

Como se mencionó anteriormente, se utilizó el lenguaje de programación C++ para desarrollar el prototipo de la Maildb API y como MRA fue utilizado *uw-imap*. Para poder interconectar el prototipo de Maildb con *uw-imap* es necesario implementar un driver para la C-Client. En este punto se genera un inconveniente, la C-Client está programada en C y el prototipo de Maildb en C++ y orientado a objetos.

Es conocido que desde C no se pueden utilizar funciones escritas en C++, debido a que los compiladores C++ distorsionan los nombres de las funciones, cosa que no sucede en C. Esta deformación de los nombres es lo que hace posible la sobrecarga de funciones en C++. Para evitar que el compilador altere los nombres de las funciones, y poder enlazar módulos escritos en C++ con módulos escritos en C, se utiliza la declaración *extern "C"* [40].

Teniendo en cuenta lo mencionado en el párrafo anterior, para solucionar el problema de interconexión entre el C-Client y Maildb, se diseñó un wrapper que actúa como mediador entre ambos. El wrapper se implementó como una librería dinámica que es cargada en tiempo de ejecución por la C-Client.
La función principal de este wrapper, es transformar los métodos de las clases existentes en el módulo de presentación, por funciones declaradas con *extern C* que puedan ser utilizadas en módulos implementados utilizando el lenguaje C.

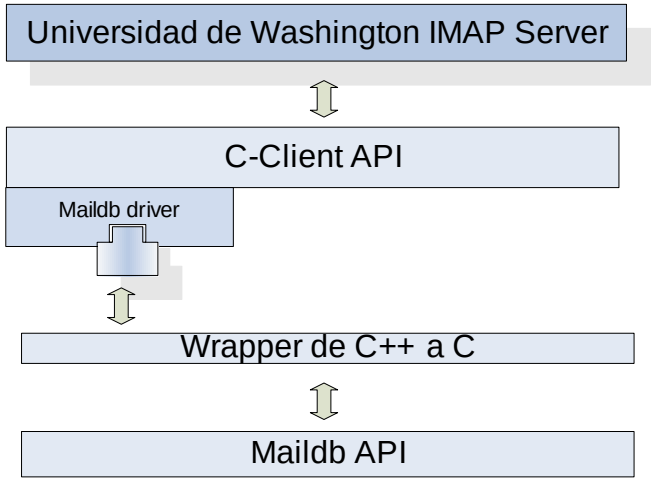


Figura IV-4 Wrapper de C++ a C

Conclusiones

Se considera que la experiencia llevada adelante en esta etapa del proyecto fue muy enriquecedora. Ofreció la posibilidad de investigar y trabajar con todos los agentes que intervienen en un servicio de e-mails. En particular conocer la implementación de un servidor IMAP, como lo es el servidor de la Universidad de Washington.

Se es consientes de que un prototipo de estas características no esta a la altura de un sistema de e-mails complejo. De todas maneras, se puede concluir que el prototipo cumple con el objetivo de proveer de un escenario realista en el cual poder realizar las pruebas de desempeño, brindándonos de esta manera una visión más objetiva sobre los resultados obtenidos.

Capítulo V

Análisis de resultados

Resumen

Este capítulo resume los resultados obtenidos de comparar nuestra solución con ciertos productos del mercado, bajo las mismas condiciones de hardware.
Los detalles completos del análisis se pueden encontrar en el "Apartado III Análisis del Desempeño de Formatos de Almacenamiento del Correo Electrónico".

Introducción

Para poder llevar adelante los análisis de desempeño de los formatos, fue necesario crear el ambiente de trabajo.
El ambiente de trabajo está compuesto por un conjunto de programas que ofrecen servicios de recuperación e-mails (MRAs) y herramientas para llevar adelante la medición de tiempos.
Además del prototipo visto en el capítulo anterior, fueron seleccionados de dentro de la gama de productos ofrecidos en el mercado, tres agentes MRA para que integraran el ambiente de análisis, éstos son:

uw-imap: Este MRA soporta los formatos *mbx* y *mbox*, pero no brinda soporte para el formato *maildir*. Este producto auspicia de agente MRA en la Facultad de Ingeniería

courier-imap: Soporta únicamente el formato *maildir,* fue seleccionado debido a que el producto anterior no soporta dicho formato.

dbmail: Es un producto de naturaleza similar al desarrollado en este proyecto y se considero interesante incluirlo en los análisis de desempeño.

Estos tres productos fueron instalados y configurados sobre el mismo equipamiento, para que estuvieran en iguales condiciones de hardware.

Las pruebas están orientadas a medir el desempeño de las operaciones que realizan los MRAs sobre el formato de almacenamiento, para poder tener mayor detalle de lo que sucede es necesario medir el desempeño de cada operación.

Se buscaron productos que ayudaran a realizar las diferentes pruebas que se tenían planificados. Dicha búsqueda fue insatisfecha, se encontraron productos interesantes, como por ejemplo *mailstone*[46], pero ninguno de ellos cumplía con el requerimiento de poder medir el desempeño de cada operación involucrada en una sesión, sino que por el contrario retornaban los tiempos insumidos por toda la sesión[7] de trabajo.

Entonces, para poder medir los tiempos insumidos en cada operación se desarrollo un pequeño cliente. Este cliente inicia una sesión conectándose al MRA utilizando el protocolo IMAP, realiza las pruebas que le indiquemos y retorna los tiempos insumidos por cada operación que se ejecute dentro de una sesión. El cliente fue desarrollado utilizando el lenguaje de programación perl.

[7] Una sesión esta compuesta por un conjunto de operaciones

Desempeño de la recuperación de e-mails

En esta etapa se mide la capacidad del formato de almacenamiento para la recuperación (total o parcial) de e-mails.

El análisis esta orientado a la realidad de nuestra casa de estudios y en base a las estadísticas se han definido tres mailbox con los siguientes tamaños:

Mailbox	Tamaño
Chico	3.5 Mb
Mediano	50 Mb
Grande	150Mb

Tabla V-1 Tamaños de mailboxes

En esta etapa se comparan los siguientes formatos:

- Basados en directorio *maildir*.
- Basados en archivos *mbox* y *mbx*.
- Basados en bases de datos *dbmail* y Maildb.

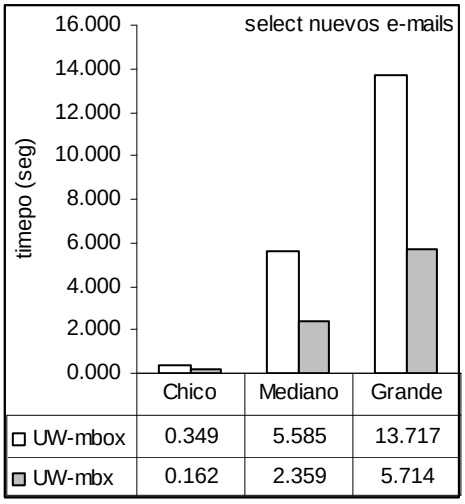
Los formatos *maildir* y *mbox* fueron elegidos por ser tradicionalmente utilizado en diversas instalaciones.
El formato *mbx* no es uno de los formatos más populares, pero fue seleccionado debido a que en la documentación del servidor de la Universidad de Washington lo consideran como un formato eficiente[49].
Dbmail fue elegido por utilizar un formato basado en bases de datos.

Las pruebas realizadas son las siguientes:

1. ***Abrir el mailbox que contiene únicamente nuevos e-mails***
2. ***Abrir el mailbox que no tiene nuevos e-mails***
3. ***Recuperar el header de todos los e-mails***
4. ***Recuperar las flags de todos los e-mails del mailbox***
5. ***Recuperar el body de todos los e-mail***
6. ***Cambiar una flag (deleted flag) de todos los e-mails.***
7. *Cambiar la flag deleted de un e-mail*
8. *Eliminar e-mails del mailbox*
    a. ***Eliminar el 25% de los e-mails***
    b. ***Eliminar el 50% de los e-mails***
    c. ***Eliminar todos los e-mails***

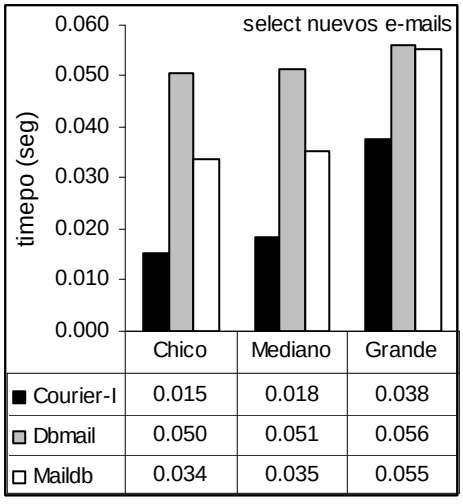
Resultados

1. Abrir el mailbox que contiene únicamente nuevos e-mails



	Chico	Mediano	Grande
UW-mbox	0.349	5.585	13.717
UW-mbx	0.162	2.359	5.714

Figura V-1 Select nuevos e-mails(a)



	Chico	Mediano	Grande
Courier-I	0.015	0.018	0.038
Dbmail	0.050	0.051	0.056
Maildb	0.034	0.035	0.055

Figura V-2 Select nuevos e-mails(b)

En las figuras V-1 y V-2 podemos observar varias cosas. Primero que nada el mejor desempeño se obtiene con el formato basado en directorios (*maildir* con el servidor *courier-imap*), seguido por los formatos que utilizan bases de datos (*dbmail* y Maildb), culminado por los formatos basados en archivos (*mbox* y *mbx*).
Estos resultados se pueden explicar de la siguiente manera. Para abrir un mailbox se utiliza la operación select del protocolo IMAP, los resultados obtenidos luego de una operación select son, entre otros, la cantidad de e-mails que existen en el mailbox y cuantos de ellos son recientes. Para obtener estos valores, los formatos basados en archivos necesariamente tienen que abrir y leer todo el mailbox contando cada e-mail, y si corresponde, agregarlo también a la cuenta de los nuevos.

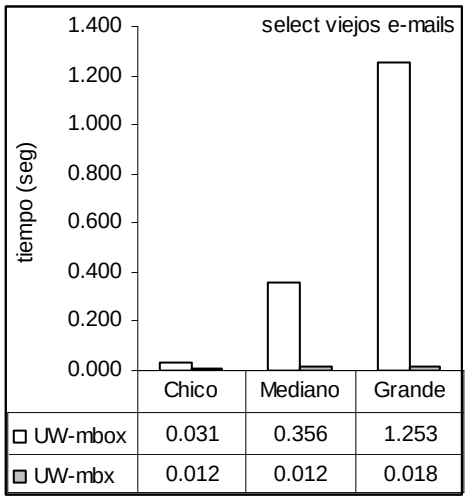
Sin embargo, en el formato basado en directorios no es necesario leer el contexto de cada e-mail, pues cada uno de éstos es almacenado en un archivo por separado dentro de la estructura de directorios. Para obtener los valores para la respuesta del comando select, simplemente alcanza con contar la cantidad de archivos que existen en el directorio e identificar cuantos son nuevos. Para el formato *maildir*, identificar cuantos e-mails son nuevos es muy simple, ya que los e-mails nuevos están almacenados en un directorio distinto (directorio new) al de los e-mails ya leídos (directorio current).

Por último, los formatos que trabajan con bases de datos, para obtener la cantidad de e-mails totales y nuevos, tienen que realizar consultas SQL sobre sus tablas.
Un factor que puede incidir sobre los tiempos de este formato, es el establecimiento de la conexión con la base de datos.

Para terminar, se agrega el siguiente comentario. Cuando se abre un mailbox que contiene e-mails nuevos, aparte de obtener los valores para responder a los resultados de la operación select, se realiza otra tarea. Esta tarea es la de asignación de UIDs (identificadores únicos) [41] a cada e-mail nuevo, la misma afecta el desempeño total de la

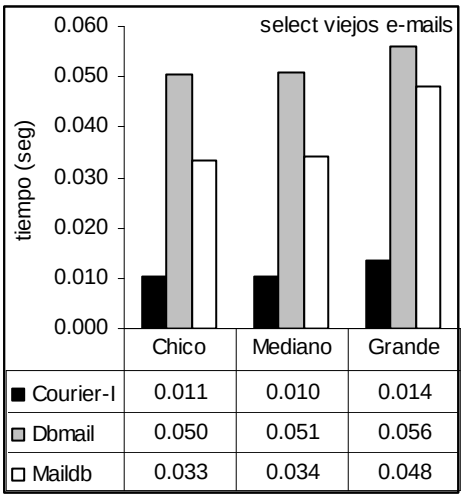
operación, en mayor o menor grado dependiendo del formato. Este último punto quedará más claro viendo los resultados de la siguiente prueba (abrir un mailbox que no contiene e-mails nuevos).

2. Abrir el mailbox que no tiene nuevos e-mails



	Chico	Mediano	Grande
UW-mbox	0.031	0.356	1.253
UW-mbx	0.012	0.012	0.018

Figura V-3 Select viejos e-mails(a)



	Chico	Mediano	Grande
Courier-I	0.011	0.010	0.014
Dbmail	0.050	0.051	0.056
Maildb	0.033	0.034	0.048

Figura V-4 Select viejos e-mails(b)

Como mencionamos en el caso anterior, select es la operación utilizada para abrir un mailbox, cualquiera sea la integración del mismo. Pero observando las figuras V-3 y V-4 podemos constatar a simple vista una mejora considerable en los tiempos insumidos por los formatos *mbox* y *mbx*, con respecto a la prueba anterior. También se constata un cambio aunque de menor magnitud en el formato *maildir*.

Si las operaciones involucradas son las mismas, el lector puede preguntarse cuál es la razón. La causa de la diferencia en el rendimiento de los formatos es la tarea de asignar los UIDs [41]k a cada e-mail nuevo.

El UID identifica a un e-mail unívocamente mientras éste exista, es decir, tiene que prevalecer entre diferentes sesiones IMAP. Este identificador es utilizado por los MUA para sincronizase con el MRA y es asignado en forma ascendente para cada e-mail nuevo. Para que el UID de un e-mail pueda prevalecer entre diferentes sesiones IMAP, es necesario persistirlo.

El formato basado en archivos, en particular el *mbox*, es el más afectado en la asignación de UIDs, la causa radica en la forma que éste persiste los UIDs. Dada la naturaleza del formato, el único mecanismo que se tiene para persistir información relacionada con un e-mail es guardarla en el mismo archivo junto al e-mail correspondiente. Para esto el servidor UW-IMAP, cuando trabaja con *mbox*, agrega a cada e-mail nuevo un X-Header que contiene el UID.
Entonces, si estamos trabajando con el formato basado en archivos y tenemos un mailbox que solamente tiene nuevos e-mails, además de recorrer el mailbox para obtener los valores para responder a la consulta select, se tiene que asignar los UIDs y luego guardar todo el mailbox. He aquí la causa del pésimo desempeño del formato al abrir un mailbox de estas características.

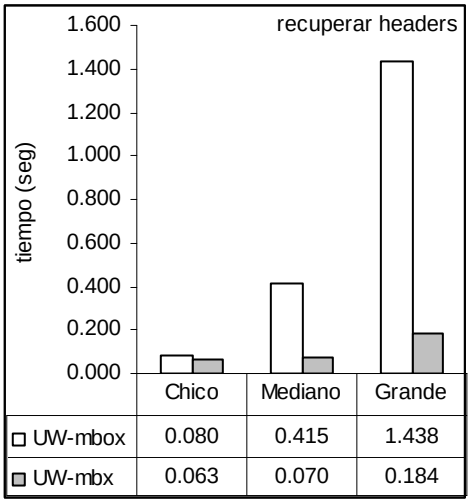
Para el formato *maildir*, el servidor *courier-imap* utiliza un archivo auxiliar en el cual guarda una asociación entre el nombre del archivo que contiene el e-mail y el UID asignado, este mecanismo es mucho más efectivo que el llevado adelante por los formatos basados en archivo.

Además de todos lo anterior, se observa que los tiempos de los formatos basados en bases de datos prácticamente no varían. Esto se debe, tanto para el prototipo de Maildb API y para *dbmail*, a que el UID es asignado al e-mail cuando éste es agregado al mailbox, y no cuando el MRA lo accede por primera vez, como en los casos anteriores.

Como nota final, podemos agregar que a pesar de que el rendimiento de los formatos basados en archivo mejora considerablemente, comparado con la prueba anterior, igual es muy ineficiente si lo comparamos con el rendimiento de los otros formatos para esta prueba.

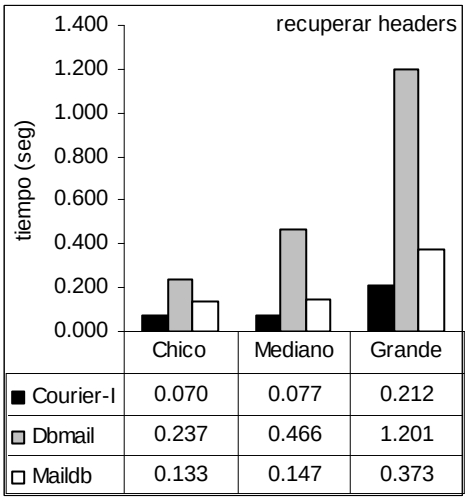
3. Recuperar el header de todos los e-mails

En este caso el mejor tiempo se obtiene para el formato *mbx*, pero los resultados para *maildir* y *maildb* están muy próximos, como lo muestran las figuras V-5 y V-6.



	Chico	Mediano	Grande
UW-mbox	0.080	0.415	1.438
UW-mbx	0.063	0.070	0.184

Figura V-5 Recuperar headers(a)



	Chico	Mediano	Grande
Courier-I	0.070	0.077	0.212
Dbmail	0.237	0.466	1.201
Maildb	0.133	0.147	0.373

Figura V-6 Recuperar headers(b)

Si comparamos los resultados del formato *mbx* con los del formato *maildir*, notamos que prácticamente comparten los mismos niveles de desempeño. Esto se debe a que en este tipo de pruebas, para acceder a información de los headers o bodys, en *maildir* es necesario abrir cada uno de los archivos que representan un e-mail y obtener la información deseada. Mientras que en el formato *mbx* presumimos[8] que se guarda información relacionada con los headers al inicio del mailbox.

Por otro lado, en el formato *mbox* es necesario recorrer todo el mailbox y obtener de cada e-mail la información solicitada.

[8] No existe mucha información de que es lo que se guarda realmente al inicio de archivo en formato binario.

Lo más atractivo de esta prueba es la diferencia que existe en el desempeño de los dos formatos basados en bases de datos.
La diferencia aquí viene dada por como esta organizada la información en las tablas. En la API Maildb los headers están almacenados en un campo de una tabla que es independiente de la tabla donde se almacenan los bodys. En *dbmail* no existe tal independencia y tanto headers como bodys están almacenados en un mismo campo dentro de una tabla, y por tanto se tiene que recuperar cada e-mail e identificar la información solicitada.

4. Recuperar las flags de todos los e-mails del mailbox

En este caso se recogieron resultados similares al caso anterior, a excepción de la mejora de los formatos *maildir* y el formato manejado por *dbmail*, en el orden de 10 veces menor. Esto se puede observar en las figuras V-7 y V-8.

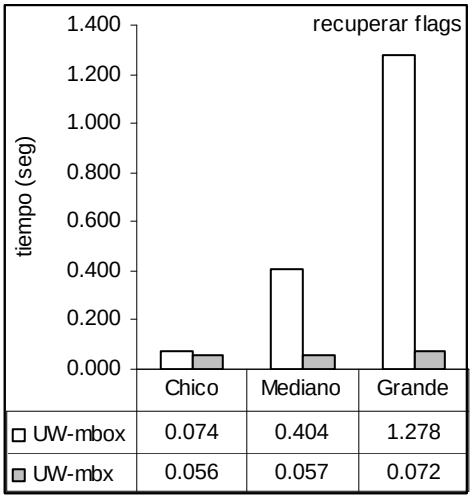


Figura V-7 Recuperar flags(a)

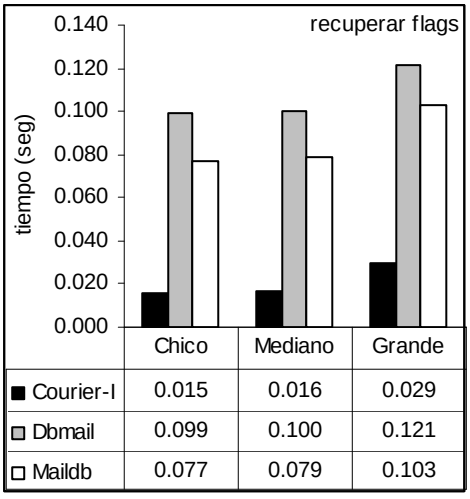


Figura V-8 Recuperar flags(b)

La razón de la mejora de ambos formatos viene dada por la forma en que almacenan las flags.
Para el formato *maildir*, las flags son almacenadas como parte del nombre del archivo que contiene el e-mail, por tanto para obtener esta información basta con ver cual es el nombre del archivo.
En el formato manejado por *dbmail* las flags son almacenadas en una tabla específica. Por tanto las consultas SQL son más rápidas. De manera similar se realiza en Maildb API.

5. Recuperar el body de todos los e-mails

En esta prueba no existen diferencia entre los diferentes formatos, ver figuras V-9 y V-10, esto se debe básicamente a dos factores. El primero de ellos es que el tiempo insumido en la transmisión de los e-mails desde el servidor MRA al MUA (utilizado para realizar las mediciones de tiempos) es tal, que hace despreciable los tiempos de recuperación de la información.
El segundo punto es que para cualquiera de los formatos es necesario recorrer todo el mailbox, pues la información solicitada es justamente toda la información de éste.

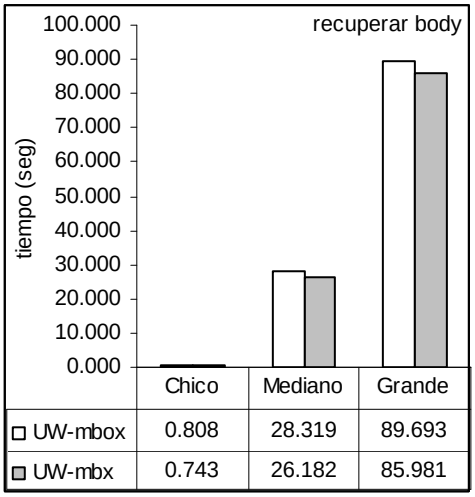


Figura V-9 Recuperar el body de los e-mails(a)

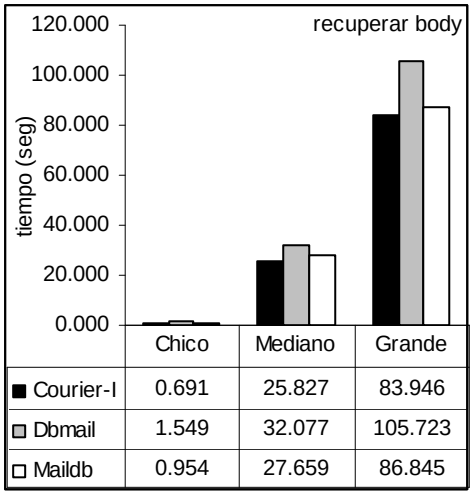


Figura V-10 Recuperar el body de los e-mails(b)

6. Cambiar una flag (deleted flag) de todos los e-mails

En este caso se marcan todos los e-mails para ser borrados (se cambia la flag deleted a positivo). Al formato *maildir* le implica renombrar todos los archivos, dado que las flags como mencionamos antes son almacenadas como parte del nombre.
Al formato *mbox* es al que le insume el mayor tiempo, esto se debe a que el servidor *uw-imap* para éste formato almacena las flags como parte de los headers, en un campo especial. Entonces es necesario recorrer todo el mailbox, asignado las flags y luego guardarlo para que las flags persistan entre sesiones.
En los formatos basados en base de datos, para cambiar las flags se tiene que actualizar campos en la base y por esta razón los valores son similares entre ellos.
Se muestran los resultados en la figuras V-11 y V12.

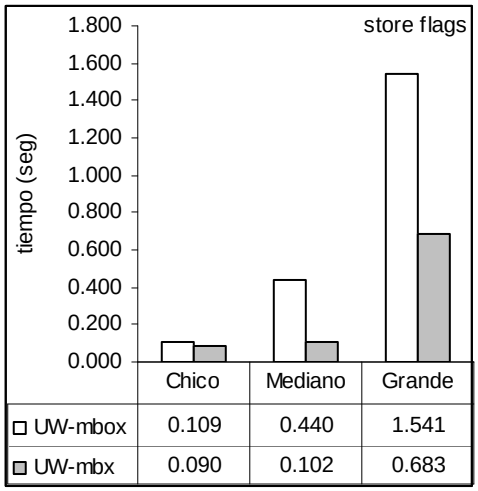


Figura V-11 Cambiar flag de todos los e-mail(a)

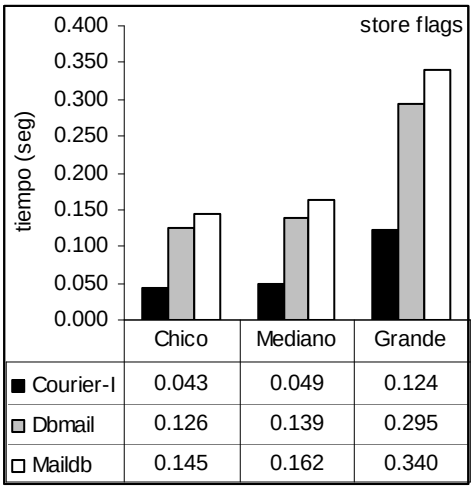
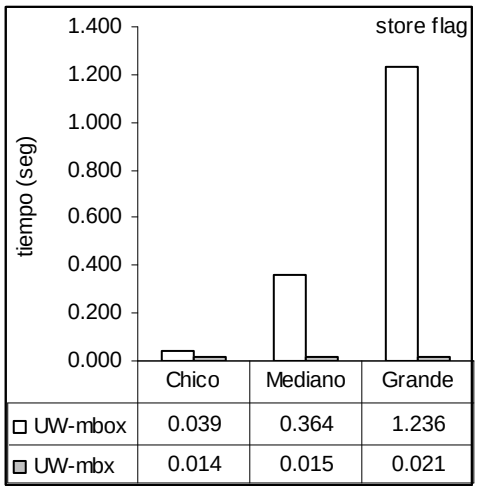


Figura V-12 Cambiar flag de todos los e-mails(b)

7. Cambiar la flag deleted de un e-mail

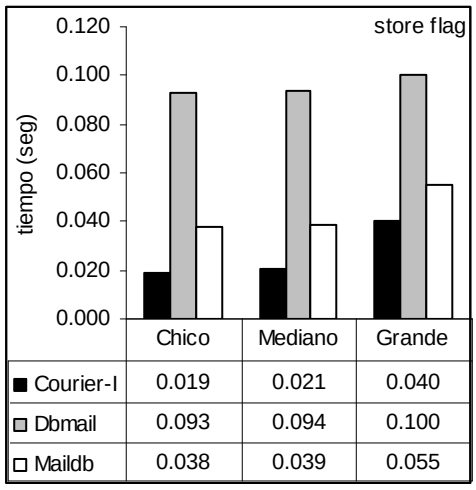
Este caso es similar al anterior pero el mejor desempeño es para el formato mbx, seguramente la información adicional que se almacena como parte del formato ayude en este proceso.
Lo curioso es la diferencia entre los resultados obtenidos para las bases de datos, si observamos el caso anterior *dbmail* obtiene un resultado levemente mejor que el de Maildb. Sin embargo en este caso el resultado es opuesto, esto quiere decir que en *dbmail* se optimiza la consulta cuando la cantidad de e-mails a modificar es mayor.

Se muestran los resultados en la figuras V-13 y V14.



	Chico	Mediano	Grande
UW-mbox	0.039	0.364	1.236
UW-mbx	0.014	0.015	0.021

Figura V-13 Cambiar flag de un e-mail(a)

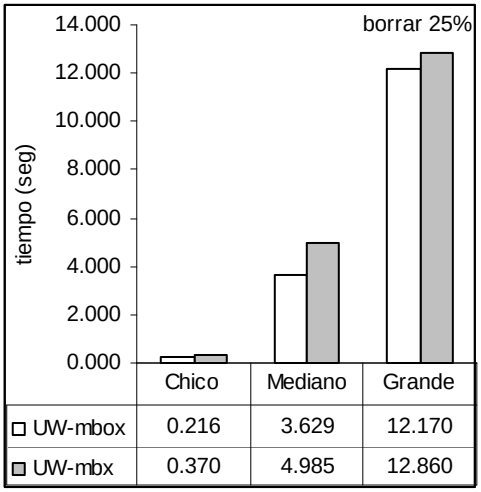


	Chico	Mediano	Grande
Courier-I	0.019	0.021	0.040
Dbmail	0.093	0.094	0.100
Maildb	0.038	0.039	0.055

Figura V-14 Cambiar flag de un e-mail(b)

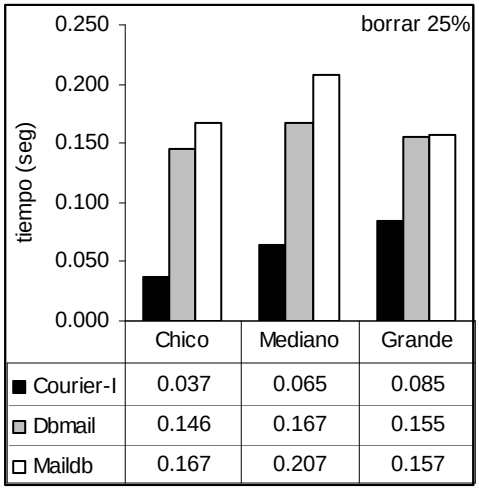
8. Eliminar e-mails del mailbox

En las siguientes 6 figuras se presentan los resultados cuantitativos de este tipo de pruebas.



	Chico	Mediano	Grande
UW-mbox	0.216	3.629	12.170
UW-mbx	0.370	4.985	12.860

Figura V-15 Borrar el 25% del mailbox(a)

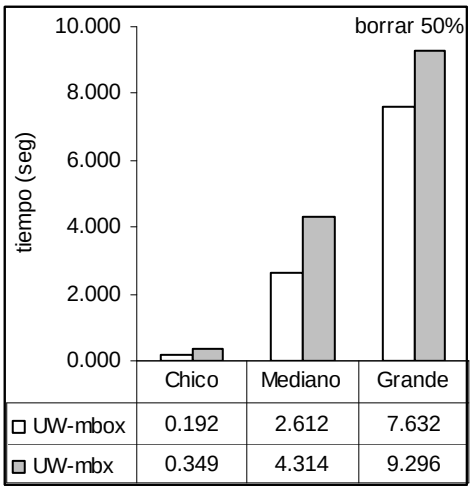


	Chico	Mediano	Grande
Courier-I	0.037	0.065	0.085
Dbmail	0.146	0.167	0.155
Maildb	0.167	0.207	0.157

Figura V-16 Borrar el 25% del mailbox(b)

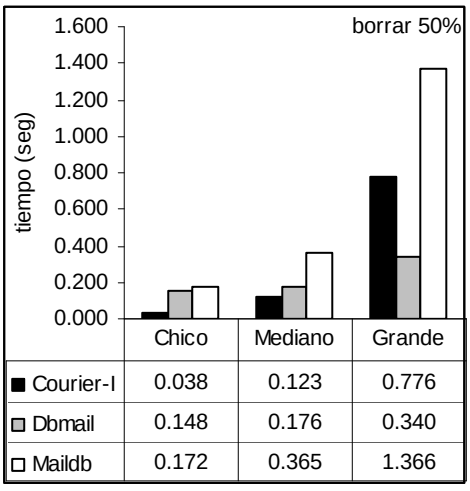
Se puede observar algunas patologías curiosas. La primera de ellas es que para el formato *mbox*, el tiempo insumido en esta prueba, es inversamente proporcional al porcentaje de e-mails que se borran. Mientras que para la API Maildb y *maildir* es directamente proporcional.

El comportamiento de *mbox* se explica de la siguiente manera, cuantos más e-mails borramos más chico es el archivo que representa el mailbox, y por ende, reescribir ese archivo lleva cada vez menos tiempo. El mejor de los casos ocurre cuando el archivo a escribir es de tamaño cero, cuando se borra el 100% de los e-mails



	Chico	Mediano	Grande
UW-mbox	0.192	2.612	7.632
UW-mbx	0.349	4.314	9.296

Figura V-17 Borrar el 50% del mailbox(a)

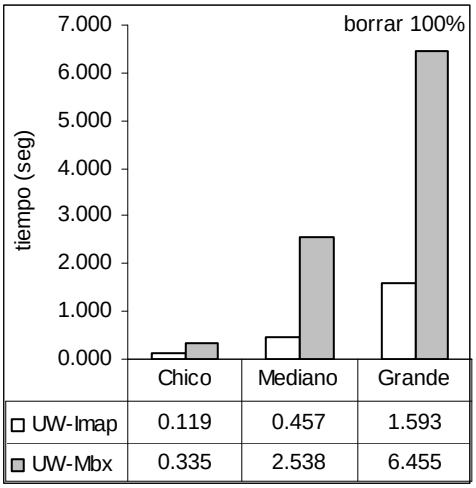


	Chico	Mediano	Grande
Courier-I	0.038	0.123	0.776
Dbmail	0.148	0.176	0.340
Maildb	0.172	0.365	1.366

Figura V-18 Borrar el 50% del mailbox(b)

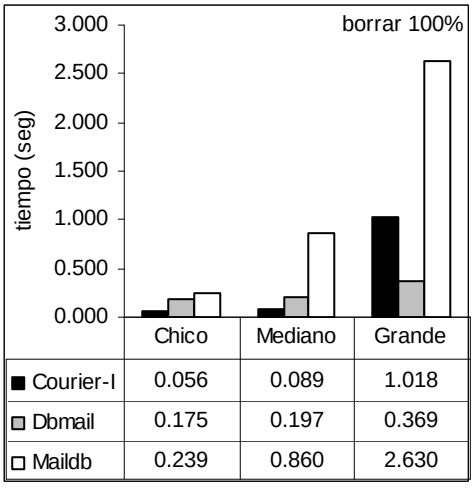
En el formato basado en bases de datos la operación de borrado de información es una operación costosa, sobre todo si se esta trabajando con índices.
Pero si observamos con detenimiento, el formato de base de datos utilizado por *dbmail* tiene un estupendo nivel de desempeño. El truco se encuentra en que en *dbmail* no se realiza el borrado propiamente dicho, sino que se realiza un borrado lógico y se provee de herramientas de mantenimiento y administración para realizar el borrado físico. Este borrado puede hacerse en horarios que afecten menos a los servicios de correo, típicamente en horas de la noche.

El formato *maildir* se encuentra afectado debido a que cuantos más e-mails se borrar mayor es la cantidad de archivos que se tienen que eliminar, y esto conduce a realizar más operaciones de entrada/salida.



	Chico	Mediano	Grande
UW-Imap	0.119	0.457	1.593
UW-Mbx	0.335	2.538	6.455

Figura V-19 Borrar el 100% del mailbox(a)



	Chico	Mediano	Grande
Courier-I	0.056	0.089	1.018
Dbmail	0.175	0.197	0.369
Maildb	0.239	0.860	2.630

Figura V-20 Borrar el 100% del mailbox(b)

Conclusiones

De los resultados obtenidos podemos observar que bajo nuestras condiciones de trabajo *maildir* es el formato que obtiene los mejores resultados en la mayoria de las operaciones. Pero no se pueden obviar los resultados obtenidos por el formato *mbx* que también demuestra tener buenas cualidades. De todas maneras la gran ventaja del formato *maildir* sobre el formato *mbx* es cuando se esta trabajando en un ambiente distribuido. En estas circunstancias, como lo anunciáramos en el capítulo II, el formato *mbx* se encuentra perjudicado por la utilización de mecanismos de bloqueo de archivos, que en *maildir* no son necesarios.

En lo que respecta al formato basado en bases de datos, si bien no superó en este tipo de pruebas al formato *maildir* demostró ser un formato a tener en cuenta. No hay que olvidar que el servidor utilizado para el formato *maildir* (*courier-imap*) es un servidor que está optimizado para este formato. En el caso del prototipo se utilizó un servidor (*uw-imap*[9]) que está pensado para brindar soporte a múltiples formatos de almacenamiento, no sacando todo el posible potencial de este formato.

Los análisis realizados permiten vislumbrar cuales son los puntos a tener en cuenta a la hora de implementar un formato basado en bases de datos, por ejemplo, uno que no se puede descuidar es el borrado de e-mails.

[9] Ver capítulo IV, sección Agente MRA.

Capítulo VI

Conclusiones y trabajos futuros

Conclusiones

El presente trabajo permitió a los autores incursionar en el área de formatos de almacenamiento de e-mails. En general los formatos se pueden clasificados en dos categorías, basados en archivo y basados en directorios. Los más populares dentro de las dos categorías son, respectivamente, *mbox* y *maildir*

En nuestras condiciones de trabajo, *maildir* superó en desempeño al formato *mbox*. El análisis realizado sobre éstos formatos, permite concluir que la principal causa del mal desempeño de *mbox*, es el tiempo insumido por la operación select[10] para procesar el mailbox, sobre todo cuando éste es de gran tamaño. De todas maneras, no hay que ignorar la existencia de algunas variantes de los formatos basados en archivos como lo es *mbx*, pese a sus puntos en contra[11]. En muchas ocasiones, el desempeño del formato *maildir* es igualado por el del formato *mbx* y en otras hasta llega a superarle.
Tampoco hay que dejar de lado el aporte de trabajos realizados por terceros, que bajo condiciones diferentes, llegan a resultados diametralmente opuestos a los de este proyecto [42]. La principal causa de esta discrepancia es la composición del mailbox que ellos utilizan, que contiene en el peor caso 21.000 e-mails. Dicha composición deja al descubierto el punto flojo de *maildir*, ya que dada la naturaleza del formato, al tener más e-mails tiene más archivos que procesar, y por ende, mayor cantidad de operaciones de entrada/salida que realizar.

En lo que respecta al formato basado en bases de datos, primero que nada, podemos destacar que la información existente, a nivel teórica y académica, es muy escueta y desactualizada [42], [44] y [45]. Por otra parte, desarrolladores de servidores de correo electrónico que trabajan en esta dirección, afirman que las bases de datos ofrecen un buen desempeño, pero no ofrecen estudios comparativos que afirmen tal aseveración.

El estudio de rendimiento realizado en el proyecto, a pesar de no estar tan completo como se pretendía[12], nos permite concluir que vale la pena seguir trabajando en esta dirección. Si bien los tiempos obtenidos no mejoran a los del formato *maildir*, están muy cercanos a ellos y no son tiempos exorbitantes, como los obtenidos para el formato *mbox*.
Además, si los comparamos los resultados obtenidos por el prototipo del proyecto, con los de un producto de similares características como lo es *dbmail*, los del prototipo son mejores en la mayoría de los casos. Con esto último, se demuestra que trabajando más en área se pueden llegar a obtener mejorados resultados.

En resumen, consideramos que se realizó un estudio particularmente interesante en lo que respecta a diferentes formatos de almacenamiento de correo electrónico (en particular los basados en bases de datos), sobre todo si se tiene en cuenta la carencia de antecedentes académicos referidos a esta área. Por lo tanto, creemos que este trabajo cumplió con su cometido, y es el puntapié inicial para futuras de investigaciones que puedan ser llevadas adelante en nuestra casa de estudio.

[10] Select es la principal operación del protocolo IMAP, y necesariamente se tiene que ejecutar para poder acceder al mailbox.
[11] El formato *mbx* utiliza fuertemente el sistema de bloqueo de archivo y esto es un punto en contra en ambientes que utilizan NFS[26].
[12] Entre otras cosas por no poder implementar en el prototipo la operación search, que suponemos es en la que obtendríamos mejor provecho de la base de datos.

Trabajos Futuros

En el área de medidas de desempeño:

En el proyecto se realizaron diversos análisis. Todos éstos tenían como cometido identificar las ventajas y desventajas de los diferentes formatos.
Otro tipo de estudio surge si miramos el problema desde la perspectiva de los administradores de sistemas. A éstos, les interesaría obtener estadísticas de cuál es el impacto causado sobre los recursos del sistema al utilizar los diferentes formatos. Por lo tanto, se necesitan realizar estudios que midan la utilización de memoria, los tiempos de proceso insumidos en cada operación, cuál es el volumen de operaciones de entrada/salida, etc.

En los últimos meses los proveedores de casillas de correo gratuitas han comenzado a brindar a sus clientes capacidades de almacenamiento del orden de los gigabytes. Es por demás interesante poder realizar mediciones similares a las del proyecto pero con mailboxes de estas características. También estudiar diferentes técnicas para poder albergar en los servidores de correo casillas de tal tamaño.

Investigar el desempeño de los diferentes formatos en un ambiente en el cual múltiples clientes acceden a los servicios concurrentemente. Una herramienta útil para simular un ambiente con estas características es *mailstore* [46].

Integrar en las pruebas servidores que manejen otro tipo de formatos no estándares, por llamarlos de alguna manera. Dentro de esta gama de servidores podemos destacar a *Cyrus IMAP* y a *Microsoft Exchange*.

En el área de desarrollo:

Dotar al prototipo de las operaciones que lamentablemente por razones de tiempo no se pudieron implementar, la más importante es la operación search que consideramos obtendría muy buenos resultados de desempeño.

Incorporar técnicas de optimización en el acceso a bases de datos para mejorar el desempeño del sistema.

Basados en la API, realizar una herramienta de administración que facilite las tareas de mantenimiento del sistema.

Se puede pensar en implementar otro prototipo que utilice frameworks desarrollados por terceros. Ejemplos de estos frameworks en el área de manejo de e-mails son: *vmime* y *gmime*. En lo que respecta a desarrollos orientados a brindar independencia de los DBMS, se encuentra un producto llamado *dbconnect*.

La API pude ser utilizada en otro tipo de aplicaciones (diferentes a MRAs), dentro de estas aplicaciones podemos encontrar: webmails y herramientas que sirvan para archivar e-mails viejos.

Hoy en día los clientes de correo cuentan con la capacidad de poder crear reglas de filtrado. Estas reglas permiten organizar automáticamente los e-mails en diferentes

mailboxes. La desventaja del mecanismo es que el filtrado lo realiza el cliente y lo por tanto, necesariamente existe transferencia de información entre el cliente y el servidor.
Si pensamos en filtros complejos basados en información que se encuentra contenida en el body de los e-mails, es necesario transmitir prácticamente todo el e-mails.
Se podría pensar entonces en dotar a la API de mecanismos para que este filtrado lo pueda realizar el servidor de correo, ahorrando de esta manera los costos implicados en la transmisión de información. Las reglas se podrían generar mediante consultas SQL avanzadas. Otro tema es el mecanismo para que el usuario le transmita el tipo de filtro al servidor.

Trabajando en la misma que mencionáramos anteriormente se puede pensar en la utilización de *triggers* (disparadores), sobre las consultas de inserción de datos , pudiendo incorporar al sistema algún mecanismo de filtrado de correo no deseado o controles de quota..

Algo interesante sería agregar a la API soporte para algunas extensiones del protocolo IMAP, como por ejemplo IMAP QUOTA[53] e IMAP ACL[54].

Bibliografía

[1]. WALES Jimmy, Wikipedia, *Multipurpose Internet Mail Extensions (MIME)* [en línea]. Ultima actualización 15:52, 22 Mar 2005. <http://en.wikipedia.org/wiki/MIME> [Consulta: 06 de abril de 2005].

[2]. *Spam Statistics* [en linea]. <http://spamlinks.net/stats.htm> [Consulta: 06 de abril de 2005].

[3]. Developed by IC&S, *DBMail software* [en línea] <http://www.dbmail.org/index.php?page=overview> [Consulta: 06 de abril de 2005].

[4]. *Oryx Mailstore* [en linea]. <http://www.oryx.com/mailstore/> [Consulta: 06 de abril de 2005].

[5]. *Email Oracle software* [en línea]. <http://www.oracle.com/technology/products/oemail/index.html> [Consulta: 06 de abril de 2005].

[6]. The Mozilla Organization, *Mozilla Suite* [en línea]. Ultima actualización 30 de diciembre de 2004. <http://www.mozilla.org/products/mozilla1.x/> [Consulta: 06 de abril de 2005].

[7]. Microsoft Corporation, *Outlook Express* software [en línea]. Ultima actualización 15 de octubre de 2003. <http://www.microsoft.com/windows/oe/> [Consulta: 06 de abril de 2005].

[8]. Developed by Computing & Communications at the University of Washington. *Program for Internet News & Email (Pine)* [en línea]. Ultima actualización 04 de abril de 2005. < http://www.washington.edu/pine/> [Consulta: 06 de abril de 2005]

[9]. *Sendmail software* [en línea]. Sitio mantenido por Sendmail Consortium <http://www.sendmail.org> [Consulta: 08 de abril de 2005].

[10]. *Qmail software* [en línea], Sitio Oficial <http://cr.yp.to/qmail.html>

[11]. VAN DEN BERG Stephen R., *Procmail software* [en línea]. http://www.procmail.org [Consulta: 08 de abril de 2005].

[12]. POZNYAKOFF Sergey, *Using mail.local* [en línea], Ultima actualización 23 de diciembre de 2004. <http://www.gnu.org/software/mailutils/manual/html_node/mailutils_34.html#SEC7> [Consulta: 08 de abril de 2005]

[13]. University of Washington. *What is IMAP?* [en línea], <http://www.imap.org/about/whatisIMAP.html> [Consulta: 08 de abril de 2005]

[14]. WALES Jimmy, Wikipedia, *Post Office Protocol (POP)* [en línea]. Ultima actualización 15:52, 22 Mar 2005. <http://en.wikipedia.org/wiki/Post_Office_Protocol> [Consulta: 08 de abril de 2005].

[15]. IMAP Information Center, *IMAP-supporting software* [en línea], Ultima actualización 27 de enero de 2005 <http://www.washington.edu/imap/> [Consulta: 08 de abril de 2005].

[16]. *Courier-IMAP software* [en línea], <http://www.courier-mta.org/imap/> [Consulta: 08 de abril de 2005]

[17]. WALES Jimmy, Wikipedia, *Simple Mail Transfer Protocol (SMTP)* [en línea]. Ultima actualización 11:12, 31 Mar 2005. <http://en.wikipedia.org/wiki/SMTP> [Consulta: 06 de abril de 2005].

[18]. KLENSIN J., *Request for Comments: 2821 "Simple Mail Transfer Protocol"* [en línea]. abril 2001. <http://www.faqs.org/rfcs/rfc2821.html> [Consulta: 08 de abril de 2005].

[19]. CRISPIN Mark, *Request for Comments: 2060 "Internet Message Access Protocol - Version 4rev1"* [en línea]. Diciembre de 1996 <http://www.faqs.org/rfcs/rfc2060.html> [Consulta: 08 de abril de 2005].

[20]. MYERS J., *Request for Comments: 1939 "Post Office Protocol - Version 3"* [en línea]. Mayo de 1996 <http://www.faqs.org/rfcs/rfc1939.html>[Consulta: 08 de abril de 2005].

[21]. *mbox* [en línea]. <http://qmail.org/qmail-manual-html/man5/mbox.html> [Consulta: 08 de abril de 2005].

[22]. BERNSTEIN Dan, *maildir - directory for incoming mail messages* [en línea], <http://qmail.org/qmail-manual-html/man5/maildir.html> [Consulta: 06 abril de 2005].

[23]. CRISPIN Mark, *Mailbox Format Characteristics* [en línea], 05 Junio de 1999. Ultima actualización 14 de febrero de 2003. <http://www.washington.edu/imap/documentation/formats.txt.html> [Consulta: 06 de abril de 2005].

[24]. DE BOYNE POLLARD Jonathan, *"mbox" is a family of several mutually incompatible mailbox formats* [en línea]. <http://homepages.tesco.net/~J.deBoynePollard/FGA/mail-mbox-formats.html> [Consulta: 06 abril de 2005].

[25]. Sleepycat Software. *Berkeley DB Data Store* [en linea] <http://www.sleepycat.com/products/data.shtml> [Consulta: 15 de abril de 2005]

[26]. STERN, Hal. *Managing NFS and NIS*. Publicado por O'reilly & Associates. 1sf ed., Junio 1991. ISBN: 0-937175-75-7

[27]. PALME J., *Request for Comments: 2076 "Common Internet Message Headers"* [en línea]. Mayo de 1996 <http://www.faqs.org/rfcs/rfc2076.html>[Consulta: 08 de abril de 2005].

[28]. FREED N. y BORENSTEIN N., *Request for Comments: 2045 "(MIME) Part One"* [en línea]. Mayo de 1996 <http://www.faqs.org/rfcs/rfc2045.html>[Consulta: 08 de abril de 2005].

[29]. FREED N. y BORENSTEIN N., *Request for Comments: 2046 "(MIME) Part Two"* [en línea]. Mayo de 1996 <http://www.faqs.org/rfcs/rfc2046.html>[Consulta: 08 de abril de 2005].

[30]. *MOORE K., Request for Comments: 2047 "(MIME) Part Three"* [en línea]. Mayo de 1996 <http://www.faqs.org/rfcs/rfc2047.html>[Consulta: 08 de abril de 2005].

[31]. CROCKER David., *Request for Comments: 822 " Standard for ARPA Internet Text Messages"* [en línea]. Mayo de 1996 <http://www.faqs.org/rfcs/rfc822.html>[Consulta: 08 de abril de 2005].

[32]. MULLET, Dianna. *Anatomy of an IMAP Session*. Managing IMAP. Editado por Mike Loukides. publicado por O'reilly & Associates. 1sf ed., septiembre 2000. p. 37-50. ISBN: 0-596-00012-X.

[33]. Oracle. *Oracle Database Product* [en línea]. <http://www.oracle.com/database/product_editions.html> [Consulta: 08 abril de 2005]

[34]. *MySQL database server software* [en línea]. <http://www.mysql.com/> [Consulta: 08 abril de 2005]

[35]. ATKINSON Kevin, *MySQL C++ API* [en línea], <http://tangentsoft.net/mysql++/> [Consulta: 06 de abril de 2005].

[36]. PostgreSQL Global Development Group. *PostgeSQL software* [en línea]. <http://www.postgresql.org/>. [Consulta: 08 de Abril de 2005]

[37]. GAMMA, Enrich. Design *Patterns CD: Elements of Reusable Object-Oriented Software Design*. Publicado por Addison-Wesley. Text of this CD corresponds to the 18th printing of *Design Patterns* 18th, 1998, ISBN 0-201-63498-8

[38]. *KDE Development Environment software* [en línea]. <http://www.kdevelop.org/> [Consulta: 08 de Abril de 2005]

[39]. CRISPIN Mark. *Documentation of c-client Functions and Interfaces* [en línea]. 19 de agosto de 1996. Ultima actualización 14 de febrero de 2003. <http://www.washington.edu/imap/documentation/internal.txt.html> [Consulta: 08 de Abril de 2005]

[40]. MILLAN Adolfo J., *Curso C++* [en línea]. Agosto de 2000. Ultima actualización 03 de abril de 2005 <http://www.zator.com/Cpp/E1_4_4.htm#enlazado%20externo> [Consulta: 05 de Abril de 2005]

[41]. MULLET, Dianna. *Messages UIDs*. Managing IMAP. Editado por Mike Loukides. publicado por O'reilly & Associates. 1sf ed., septiembre 2000. p. 41. ISBN: 0-596-00012-X.

[42]. ELPRIN, Nicholas & PARNO Bryan, *An Analysis of Database-Driven Mail server* [en línea], Computer Science Group, Harvard University, Cambrige Massachussets. <http://www.usenix.org/events/lisa03/tech/elprin.html> [Consulta: 17 de Abril de 2005] Ultimo cambio 7 de noviembre de 2003.

[43]. VARSHAVCHIK Sam, *Benchmarking mbox versus maildir*, Courier. <http://www.courier-mta.org/mbox-vs-maildir/> [Consulta: 17 de Abril de 2005]

[44]. LEAGUE Christopher, *Electronic Mail Database System* [en línea]. 15 de mayo de 2000. Ultima actualización 25 de Julio de 2003. <http://contrapunctus.net/league/teaching/meba.pdf> [Consulta 07 de Abril de 2005]

[45]. KNET J.,TERRY D. y Orr W-S. *Browsing Electronic Mail: Experiences Interfacing a Mail System to a DBMS* [en línea]. Computer Science Laboratory, Xerox Palo Alto Research Center. Noviembre de 1989. <http://www.parc.xerox.com/about/history/pub-historical.html> [Consulta: 07 de Abril de 2005]

[46]. CHRISTIAN Dan, *Mailstone* [en línea]. Ultima actualización 13 setiembre 2004 <http://www.mozilla.org/projects/mstone/> [Consulta: 08 de Abril de 2005]

[47]. *GNU General Public License* [en línea]. 1 de abril de 1989. Ultima actualización 08 de abril de 2004. <http://www.gnu.org/copyleft/gpl.html>. [Consulta: 08 de Abril de 2005]

[48]. INGRAM Johnathan. *Dbconnect API* [en línea], <http://dbconnect.sourceforge.net/> [Consulta: 17 de Abril de 2005]

[49]. CRISPIN Mark, *C-Client Driver Characteristics* (VI Driver Comparison) [en línea], 05 Junio de 1999. Ultima actualización 14 de febrero de 2003. <http://www.washington.edu/imap/documentation/drivers.txt.html > [Consulta: 06 de abril de 2005].

[50]. VINCENT Richard, *vmime* [en línea],<http://vmime.sourceforge.net/> [Consulta: 16 de abril de 2005].

[51]. STEDFAST Jeffrey, *gmime* [en línea], <http://spruce.sourceforge.net/gmime/> [Consulta: 16 de abril de 2005].

[52]. DINH Viet Hoa, *libetpan* [en línea], <http://libetpan.sourceforge.net/>, Ultima actualización 30 de diciembre 2004, [Consulta: 16 de abril de 2005].

[53]. MYERS J., *Request for Comments: 2087 "IMAP4 QUOTA extension"* [en línea]. Enero de 1997 < http://www.faqs.org/rfcs/rfc2087.html>[Consulta: 08 de abril de 2005]

[54]. MYERS J., *Request for Comments: 2087 "IMAP4 ACL extension"* [en línea]. Enero de 1997 < http://www.faqs.org/rfcs/rfc2086.html>[Consulta: 08 de abril de 2005]