Internet Messaging Frameworks

by J. von Känel
J. S. Givler
B. Leiba
W. Segmuller

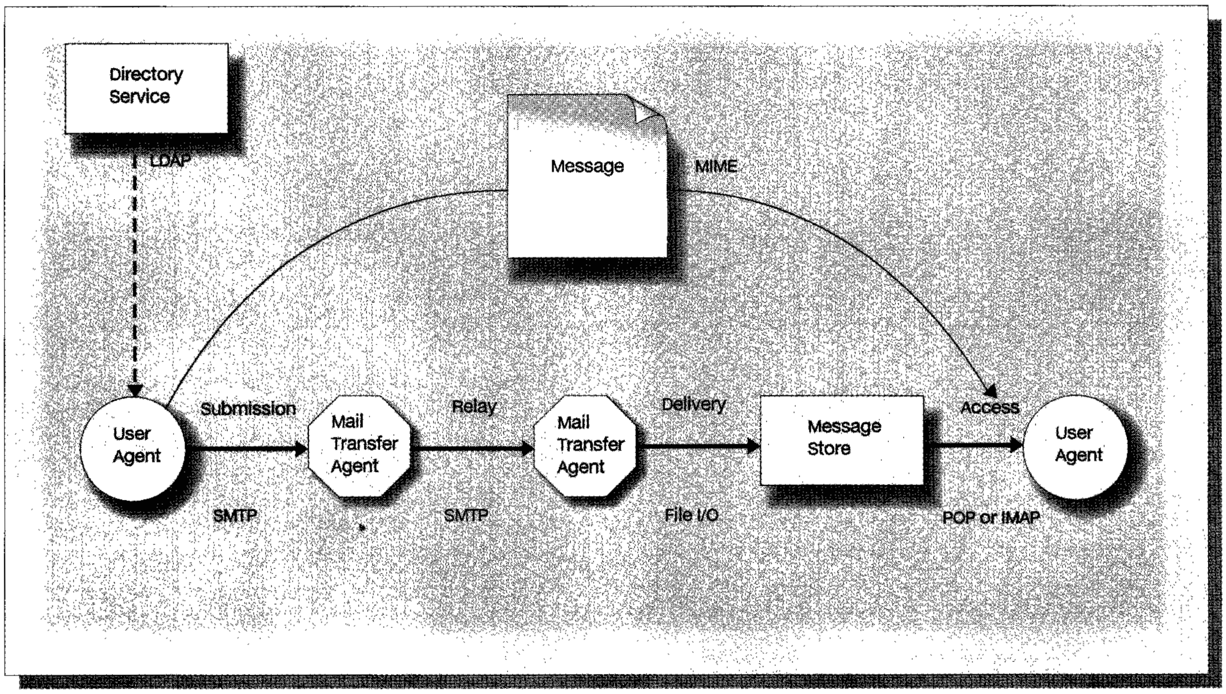
Electronic mail (e-mail) has become an important tool for companies to use to conduct their businesses. With the introduction of the World Wide Web, awareness of the existence of the Internet has exponentially increased over the last two years, and people are starting to realize that there is more to the Internet than just the Web. Companies are expanding their use of e-mail from internal to external. But the large set of proprietary, noninteroperable e-mail systems make this more of a trip through a jungle than a drive along the information highway. Most approaches to overcome the connectivity problems use gateways to convert between the proprietary format and the Internet standards. These conversions are lossy at best; hence, most proprietary system vendors are revamping their systems to base them on Internet standards. This paper summarizes the current state of the most important Internet standards related to e-mail and the general state of proprietary e-mail systems. It then introduces a set of technologies we developed to solve the complex problem of evolving from proprietary to Internet-standards-based e-mail systems. We have structured these technologies into Internet Messaging Frameworks.

Company electronic mail (e-mail), a mere novelty a few years ago, is a mission-critical part of the company infrastructure today. Proprietary e-mail systems, like cc:Mail** or Lotus Notes**, have evolved over time, and users appreciate their nice user interfaces, rich functionality, security, receipt notifications, and a multitude of other features. With the World Wide Web giving easy access to a free-flowing information exchange, more and more businesses want to move from the one-way Web to bidirectional e-mail exchange with their customers and suppliers. The first step invariably involves setting up a gateway to connect the proprietary mail system to the Internet—and then reality sets in. A lot of things that used to work are not working any longer or not working quite right. Not all mail gets delivered, return receipts are a gamble, some of the mail coming from the Internet gets garbled into many parts, and puzzling out what the sender intended is difficult. So what will happen next?

Obviously, some standard way to hook everything together is needed. The Open Systems Interconnection (OSI) X.400 standard was believed to be such a standard. However the design became overly complex, and its implementations never interoperated well. The Internet was built to hook together a vast number of heterogeneous networks and was designed for commonality and simplicity long before X.400 was in place. Today the Internet is the world's largest network, consisting of a set of interconnected networks spanning the whole planet.

Due to the problems in connecting proprietary systems to the Internet via gateways satisfactorily, most e-mail system vendors are abandoning their proprietary approaches and are migrating their systems to become Internet-standards-based. This is done by adapting their proprietary mail model to the Inter-

Figure 1 Internet message transport overview



net mail model and by eliminating the need for gateways. Often this cannot be achieved in a single release, but has to be staged over several releases to achieve a more or less smooth migration for their customers.

The body of this paper has two distinct parts. The first part gives an overview of the most important e-mail standards of the Internet and the general technological state of proprietary e-mail systems, providing a frame of reference for the second part. The second part of the paper introduces a set of technologies that we have developed to help build new Internet e-mail clients and servers, as well as to allow existing, proprietary clients and servers to be easily adapted for Internet standards compliance. In our conclusions we outline how these technologies have been used to build the Lotus Java**-based eSuite Workplace** e-mail client and to migrate cc:Mail clients to become Lotus Mail clients.

The Internet and electronic mail

The Internet has been designed and built to connect a large number of heterogeneous systems in an interoperable way. The basic infrastructure of Internet e-mail can be described as a set of synergistic standards describing message transport, message formats, message access, security, and directory services. The Internet Engineering Task Force (IETF) publishes specifications of Internet-standard protocols and formats, which are agreed upon by the IETF participants. These standards, called "RFCs" (requests for comments), allow systems produced by different designers to cooperate with each other and exchange information, including e-mail (see Figure 1).

The *message transport* model describes how a message travels from the originator to the recipient. In general, a program used to display and create messages is called a *user agent* (UA). The originating user agent submits the message to the *mail transfer agent* (MTA). Depending upon where the recipient user agent is in the network topology, the message might be relayed one or more times. Once the message reaches the destination MTA it is delivered into the *message store*. The recipient user agent can then access the message for display and further user actions. In the Internet the *Simple Mail Transfer Protocol* (SMTP) is used for the submission and relay of mes-

Figure 2 RFC822 message

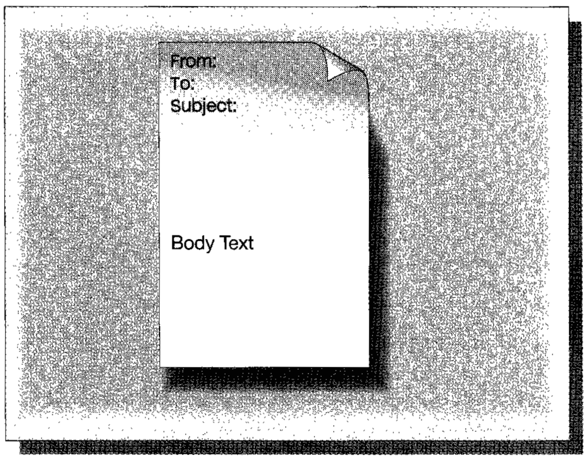
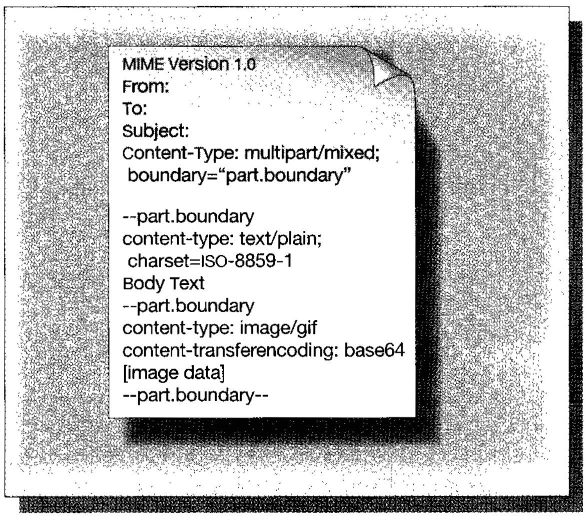


Figure 3 MIME message



```
MIME Version 1.0
From:
To:
Subject:
Content-Type: multipart/mixed;
 boundary="part.boundary"

--part.boundary
content-type: text/plain;
 charset=ISO-8859-1
Body Text
--part.boundary
content-type: image/gif
content-transferencoding: base64
[image data]
--part.boundary--
```

sages. Usually simple file I/O is used to deliver the messages into the message store. Either POP3 or IMAP4 is used to access these message stores.

The original *message format* used on the Internet is the basic RFC822 message format. It is structured similarly to a memo in the physical world, consisting of a header and a body. (See Figure 2.) The message header describes the sender, the recipient, the subject, the date, and other such items. The body of the message has no defined structure; it is just text. Both header and body can contain only 7-bit US-ASCII (United States-American National Standard Code for Information Interchange) data. The 7-bit US-ASCII restriction results in limitations in other countries, where the character set cannot be described in US-ASCII.

The *MIME* format (Figure 3) is an extension to RFC822, used to bring structure to the body and to allow for the transport of complex, multipart messages containing text, images, audio, video, and other binary attachments. It also removes the character-set limitations, allowing character sets other than US-ASCII, including the multibyte character sets needed to represent some Asian languages.

MIME adds a few new fields, such as the MIME-version field, to the header to distinguish the MIME messages from plain RFC822 messages. The content-type field describes the data type of the body. Seven basic MIME types have been defined: text, image, audio, video, application, message, and multipart. Each type has several subtypes defined: text/plain and text/html are two examples of text subtypes. MIME also introduces "transfer encodings" to allow binary data to travel as part of a message after being encoded into ASCII characters in a standard way. MIME introduces definitions to allow character sets other than US-ASCII to be encoded as part of the header text fields or the body.

The MIME message model is a "recursive parts" model: the body is a part, and each part can contain other parts. This recursiveness is very powerful since some parts can influence the representation of their subparts. For example a multipart/mixed part contains a series of, not necessarily related, subparts, with the intent that all subparts be presented to the user. In contrast, a multipart/alternative part contains a series of semantically equivalent subparts (for example, an image and a textual description of the image), only one of which should be displayed by the user agent. This multipart/alternative form is quite commonly used by browsers to include a plain text and an HTML (HyperText Markup Language) version of the message.

Once messages have been delivered into the mailbox of the recipient's message store, the recipient needs *message access* methods to retrieve and work with the messages. Currently there are two standard ways to access message stores.

POP3 is the simple Post Office Protocol (version 3). It treats the message store as a single in-box. The user agent can retrieve and delete messages from this in-box. Once messages are retrieved and deleted from the POP3 server, it is the user agent's responsibility, if necessary, to retain messages in some local message store. While a POP3 client can leave mail on the server (by not deleting it), the POP3 protocol lacks mechanisms to categorize, file, or search the mail, so the POP3 server message store can quickly become unmanageable. Also, most large-scale POP3 servers enforce a storage limit, refusing to accept new mail for a user whose limit has been exceeded. Thus, the POP3 model strongly encourages the complete transfer of mail to the client, where a well-designed client can provide many more capabilities to the user. This has the advantage that the communication with the server is simple, but it has the disadvantage that the user cannot conveniently use more than one computer to read mail: the mail remains on whichever computer the user reads it.

IMAP4, the Internet Mail Access Protocol (version 4), is a newer access protocol that defines a much richer message store, allowing mail to be stored in multiple mailboxes. A rich set of message and mailbox manipulation functions exist. While a POP3 message can be handled only as a single block, IMAP4 allows access to individual MIME parts. Provisions exist to allow message stores to be replicated to a local store (and resynchronized later) for the mobile user. The IMAP4 model, in contrast to the POP3 model, involves storing mail on the server, where it may be accessed by any client, and using the client's storage only for caching messages for efficiency or for traveling.

POP3 is currently widely deployed by Internet Service Providers (ISPs) for access to users' mail. Because of its simplicity, it will probably remain the major access protocol for the casual mail user for quite some time. IMAP4 is not yet widely deployed, but due to its functionality, which is more suited to the traveling business user, it will increase its deployment throughout the business community over the next few years.

The set of standards described so far allows messages to be transmitted through the Internet, but only "in the clear." There is no inherent *message security* built into them. In fact, it is relatively simple to send messages that appear to come from someone else. To conduct business on the Internet, features such as authentication and encryption are needed to make message transmission secure. *Authentication* allows messages to be signed, so the recipient can confirm that the sender is the person claimed. *Encryption* allows data to be sent in such a fashion that only a recipient with a key can decrypt the data.

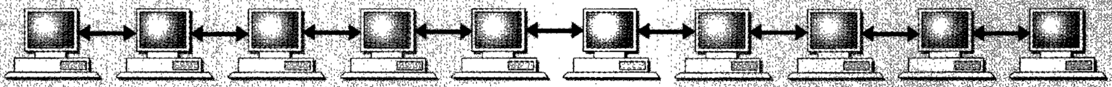
The security schema most widely used today on the Internet is *PGP* (pretty good privacy). It relies on a "web of trust" for the publication of keys. This web-of-trust model is one of PGP's major strengths in the self-governing Internet society. However, it is not well accepted in the business community, which would like a hierarchical trust model, with signing authorities to guarantee keys. *S/MIME* is currently under discussion by the IETF as an alternative security mechanism for e-mail.

While *directory services* have their own niche in the set of Internet standards, they are central to many applications. For e-mail they are needed to access user information, such as a given user's e-mail address. *LDAP*, the Lightweight Directory Access Protocol, is the standard describing how to access directory data. Directory services will play an even greater role for storing and accessing public keys to enable secure messaging. While users can remember a large number of e-mail addresses or even keep track of them in personal address books, the same cannot be said for keys, which are lengthy, seemingly random character strings.

The state of proprietary e-mail systems. Most proprietary systems have been developed for a homogeneous group of users on a single network. They typically have a large set of features allowing the creation and manipulation of compound documents. Their delivery systems often support guaranteed deliveries and receipt notifications. Additional integrated functions for calendars and schedules are not uncommon. On the other hand, they often do not scale well to large user communities, because they were developed for a small, homogeneous domain. They cannot exchange mail with other systems except through specially designed gateways, which lose information in the process of converting between mail formats.

The mail format in proprietary systems is often the "cover letter and attachments" model from the physical world of mail. There is typically a special text part called "the message" and a set of attachments. Often the number of possible attachments is very limited—it can be as few as one, or perhaps as many as twenty. To integrate these mail systems with the Internet, the gateways have to perform a conversion between the Internet format and the proprietary for-

INTERNET E-MAIL STANDARDS ACTIVITIES



A selection of the most important standards and standards activities with regard to e-mail are listed here. While for most of these a subjective statement is made on their pervasiveness, no hard data exist that show, for example, how many POP or IMAP seats have been deployed.

SMTP (RFC821, standard)

The Simple Mail Transfer Protocol is the fundamental standard defining how to transfer (ASCII) data from one point to another. It is currently undergoing a general revision to bring all minor revisions of RFC821 together into one concise RFC. SMTP is universally deployed.

RFC822 (RFC822, standard)

The "Standard for the format of ARPA (Advanced Research Projects Agency) Internet text messages" is commonly called RFC822. It defines the basic format of a mail message. It is currently undergoing a general revision as part of the IETF (Internet Engineering Task Force) effort to streamline and clarify the various mail-related standards. RFC822 is universally deployed.

MIME (RFC2045-2049, draft standard)

The Multipurpose Internet Mail Extensions define how complex, multipart messages with binary attachments are structured.They also define how messages with other than US-ASCII character sets have to be encoded. MIME is widely deployed.

MHTML (RFC2110-2112, proposed standard)

The encoding of and use of HTML documents as the compound document format is the current emerging trend in e-mail. It gives e-mail the rich formatting capabilities that are now available for Web pages, allowing a sender to use things such as highlighting, tables, and imbedded images in mail messages. MHTML (MIME-HTML) deployment is emerging.

Receipt (draft, proposed standard expected fall 1997)

Users often want to confirm receipt of e-mail messages, a function that has generally been available in proprietary mail systems but not on the Internet. Receipt deployment is emerging.

PGP (RFC2015, proposed standard)

The "pretty good privacy" security extensions are currently the most widely deployed security schema for e-mail.

S/MIME (draft, no proposed standards yet)

S/MIME (Secure MIME), as defined by RSA Data Security, Inc., has some minimal deployment. Current discussions in the IETF about whether S/MIME should become an Internet standard are stalled. Most likely it will be published as an informational RFC and become a *de facto* alternative to PGP.

POP3 (RFC1939, draft standard)

The Post Office Protocol defines a simple message store consisting of one mailbox and two simple operations: reading and deleting mail from the mailbox. POP3 is widely deployed as the preferred mail access protocol for ISPs.

IMAP4 (RFC2060, proposed standard)

The Internet Mail Access Protocol defines a more complex message store, which allows multiple mailboxes and a rich set of functions on this message store. IMAP4 is not yet widely deployed.

LDAP (RFC1777, draft standard)

The Lightweight Directory Access Protocol is used to store and find information, such as the e-mail address of a person. LDAP has some deployment.

A much more detailed but concise list of e-mail-related standards activities can be found at the Internet Mail Consortium Web pages: http://www.imc.org.

mat. Their biggest problem in this area is in handling the recursive parts described earlier. It has become increasingly common for an incoming message from the Internet to have recursive parts, either because the sender's user agent provided alternatives (HTML and plain text, generally) or because the message contains an embedded message complete with its own parts (a forwarded message, for example). This recursive relationship between the parts is usually lost in the gateway: often the parts will just be converted into a linear set of attachments and the user has to guess how they fit together. It is also possible that there is no text part in an Internet message—perhaps just an image or a sound clip. This will typically generate an empty message with some attachments, and the empty message may be confusing to the recipient.

Large companies often have several different such e-mail systems. Management, administration, and interoperability is difficult and expensive. As e-mail becomes critical to the business, such companies need to install a plethora of gateways to connect all these systems together. Often the only feasible solution is to create an SMTP-based "backbone" into which all proprietary systems connect via gateways. The results can be very frustrating, due to the loss of information in the gateways.

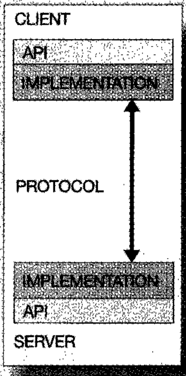
Smaller companies usually have just one proprietary e-mail system, and they may be happy with it for some time. But once they find it necessary to communicate with the Internet, they must decide how to do it. It is then a question of whether they should install a gateway to the Internet, or switch over to an Internet solution completely.

For builders of proprietary e-mail systems, there are many questions and problems. To survive, they must either build gateways between the Internet and their system (short-term solution), or redesign their systems to use Internet standards natively (long-term solution). More likely, they will have to do both: build gateways to retain their current customers and provide for migration to their Internet native solution later.

The first difficulty is the format problem just described. More often than not, builders must completely redesign their graphical user interfaces (to be able to display and create complex messages in the recursive Internet style) and their storage mechanism (to store MIME data rather than "cover-letter-and-attachment mail").

APIs VERSUS PROTOCOLS

Many vendors publish a set of APIs (application programming interfaces that represent the procedure calls and parameters for an application to invoke a function) to their mail systems and then call this an open system. In the Internet arena APIs are not considered open, because every client/server communication requires a protocol. The protocol specifies how data travel "across the wire" between the client and the server.



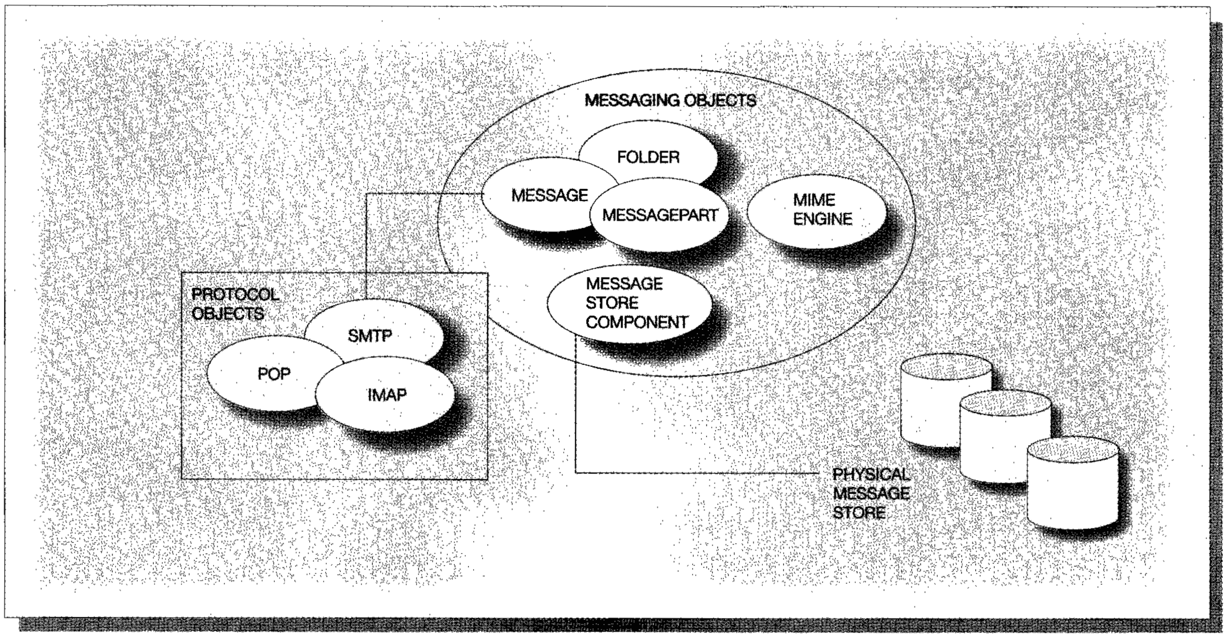
A high-level architectural view is illustrated. The client application uses an API to gain access to the server. The implementation of the API uses a protocol to get the data from the client computer over to the server computer. The data end up in the implementation layer on the server. The server application uses another API to receive the data. The response to the client's request then traverses the layers in reverse.

In most cases the picture is not really symmetrical. The vendor publishes the API on the client side. On the server side, everything is often just collapsed into one monolithic implementation or hidden in a lower layer of the operating system or middleware. The protocol used by the vendor is typically proprietary and undocumented: only the implementation shows what kind of protocol was used. The vendor usually does not provide the implementation code, the protocol is often not well-defined, and the details of the protocol usually change from release to release, making direct use of the client/server protocol very risky.

For true interoperability, as required in the Internet arena, the protocol defining the packet formats on the "wire" has to be standardized. This is the only way to be completely unbiased about operating systems and programming languages, allowing interoperability between heterogeneous systems. A completely interoperable API would have to be defined in all programming languages and be implemented on each operating system, an almost insurmountable task.

The IETF publishes protocols, not APIs, as standards. In some cases APIs for reference implementations are published, but the APIs are never "the standard."

Figure 4 Frameworks overview



Another significant problem is in the area of APIs. Many proprietary mail systems claimed to be open, where "open" was defined as having a published API. However, this is not what the Internet community considers open. An API usually just provides access to a "black-box" implementation of a proprietary protocol, and so one cannot really write another interoperable client or server: the black box must be reverse-engineered to make it truly work. This is often true for the vendors themselves, because the protocol has never been documented (other than in the source code behind the API implementation). They now have to change system architecture to base it on the Internet standards protocols. The APIs themselves become less useful than they seemed at first.

Internet Messaging Frameworks

The job of architecting, designing, and implementing e-mail clients and servers based on Internet standards is by no means trivial. The main focus of our research has been to find ways to simplify the work of the implementers of clients and servers for such systems. This was achieved by creating the Internet Messaging Frameworks, which encapsulate a necessary and sufficient set of objects to express an abstract notion of Internet e-mail and its associated protocols.

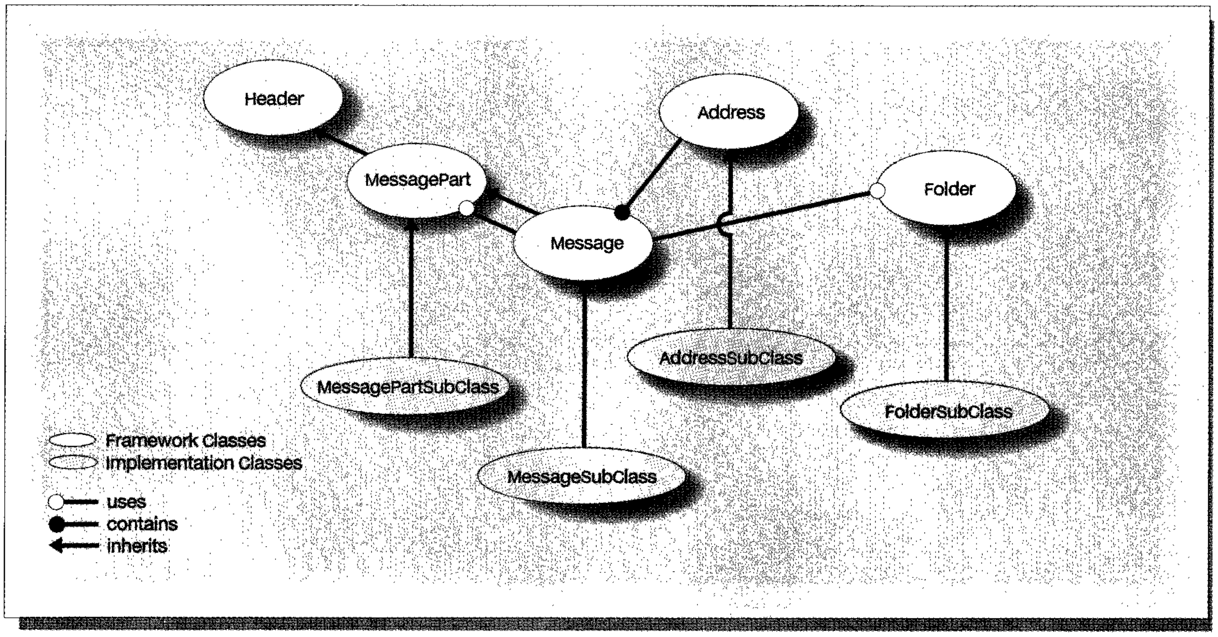
We have identified a set of high-level abstractions, which are used to implement both clients and servers. The architectural overview can be seen in Figure 4.

The *messaging objects* are the fundamental framework. They abstract the notions of message, message parts, folders, and e-mail recipients. The classes for messaging objects must be subclassed for any particular *message store* implementation. A default implementation of a memory message store, required for a program to work with messaging objects, is provided as part of the base framework implementation.

The *MIME engine* is a generic parser/generator framework. It efficiently parses a MIME stream into any object model. The messaging objects create specializations of the MIME engine to convert between the MIME stream and the messaging objects.

Protocol objects are different for clients and servers. On the client side they issue requests to the server on behalf of the messaging objects. On the server

Figure 5 Classes for messaging objects



they execute the request from the client on the messaging objects.

The extent to which we have been able to separate out components into related but independent parts goes far beyond typical approaches, which merge all the MIME support in with the message and message-part implementation of their mail model. Our approach of separating the work into the distinct pieces of protocol, internal representation (the messaging objects), MIME parsing and generating, and backend storage gives the implementer enormous flexibility. Since the protocol component is isolated, a client or server implemented with this framework can easily be made to operate with many different protocols (both standard and proprietary) by providing alternative protocol implementations. Similarly, the isolation of the message store backend makes it easy to implement multiple backends, allowing the same server to store mail in many different databases and file systems. By isolating the MIME engine, we have a single, robust component, where all MIME-related operations are encapsulated, and that is very easy to maintain, debug, extend, and enhance.

In the next few sections, the components of the Internet Messaging Frameworks and their uses in implementing clients and servers are described in much finer detail.

Messaging objects. The messaging objects (Figure 5) are the core of the framework, used by both clients and servers to model MIME messages and IMAP4 folders. The base framework contains a memory implementation used by programs to manipulate these objects. For permanent storage, the message-store interfaces of the framework must be specialized for any particular physical message store. For example, to store mail in cc:Mail's DB8 format, a DB8 interface must be implemented.

There are messaging objects to represent folders, messages, message parts, and e-mail addresses. A *folder*, also called a mailbox, is a collection of messages and (other) folders. Each message is uniquely identified in the folder. The *Folder* class is an abstract class, providing an interface for creating, deleting, retrieving, and searching entries in a folder. The *Message* class is used for objects that represent messages in the folder.

The *Header* class is an abstract class. It provides an interface to set and query the unstructured and optional fields of a message or message-part header.

A *message part* consists of a header and a body. The header contains information describing the contents of the body. The body may be a stream of data, a container of nested parts, or an embedded message. There are four distinct categories of methods for a message-part object. The first and most important group consists of the header methods, which, among other things, allow the setting and querying of the content type. The remaining three categories of methods are for each of the different content types of the body.

The *MessagePart* class is used for objects that represent a message part. The header portion is derived from the Header class, augmented with additional methods to support the content-type and content-disposition header fields. For the body portion, three different categories of types are supported: atomic parts (text/*, image/*, etc.), recursive parts (multipart/*), and embedded messages (message/rfc822). For atomic parts there are accessor methods to the data stream. For recursive parts the methods allow the creation, enumeration, and manipulation of nested parts. For embedded messages there are methods to set and get the embedded message, which is represented by a note object.

The *Address* class is used for objects that represent a recipient or a list of recipients. An address object stores the display name, e-mail address, and comment, along with other information, such as whether the message must be sent to this address or whether the message has already been sent to this address. When an address object is a list of recipients, the object stores an ordered list of address objects. If it is a group, then a group name is stored, as well as an indication of whether the list should be expanded or just the group name included when the message is sent.

A *message* is represented as a message part. This is an important aspect of implementing recursive parts: since a message part may itself be a complete message (MIME type message/rfc822), by representing all messages as message parts we ensure that the messaging objects will behave properly for embedded messages, with no extra work required. Our Message class is a subclass of the MessagePart class and extends its interface with methods that deal with properties of the message (such as the list of recipients).

The MIME engine. The MIME engine is a generic module that simplifies the handling of MIME-encoded data. It presently encapsulates most of the IETF e-mail specifications found in RFCs 822 (e-mail), 1468 (ISO-2022-JP), 1641 (Unicode), 1642 (UTF-7), 1806 (Content-Disposition), 2044 (UTF-8), 2045-2049 (MIME), a bit of RFC 1138 (X.400), and a draft proposal for acknowledgments (receipts). Other emerging IETF specifications are being tracked, covering issues such as acknowledgments, encryption, authentication, and internationalization of character sets. Our MIME engine is designed in such a way that it does not enforce any particular mail-model implementation. The messaging objects introduced in the previous section are one example of a possible mail model.

Within IBM, there are applications that make use of this parser technology but do not use the messaging objects; instead they specialize the parser framework to fit into their own model. This approach, of completely separating the MIME engine from the rest of the system, is in contrast to the usual implementation that incorporates the knowledge of MIME and related message-format standards throughout. With this unique separation it is easier and less error-prone to add support for new and emerging standards (such as the receipts proposal described earlier).

The MIME engine consists of two major pieces: the parser (for inbound messages) and the generator (for outbound messages). The MIME parser and generator are usually compiled and linked into a single module. The engine is thread-safe and does not require multiple threads for its own implementation. The engine's storage requirements are proportional to the complexity of the message and not to the size of the message's body or attachments.

The parser and generator interfaces contain a few classes that are subclassed by the client. These interfaces are used to pass both information and program control back and forth between the engine and various functions in the client.

The MIME parser. It is the responsibility of the parser to take an incoming MIME message, dissect it into its component parts, and inform the client program of all nontrivial components. The parser handles line unfolding, transfer decoding, and (optionally) character-set conversions of text parts.

The design philosophy behind the parser is to correct as many errors as possible when parsing messages, since there are a number of "almost-legal" MIME messages floating around the Internet. This error correction makes it impossible to use table-driven parsing approaches via lex and yacc, methods commonly used in other MIME parsers. We have found, however, that there is an elegant object-oriented approach to this problem, and we have encapsulated that approach in our MIME parsing engine.

The client provides the parser with an input-stream object, which contains the incoming message, and one or more output-stream objects, into which the parser will place the body of the message and its attachments. Additionally, optional hooks are available for the parser to report to the client the values of the MIME header fields (for example, "To:" or "Subject:") in the various parts of the message.

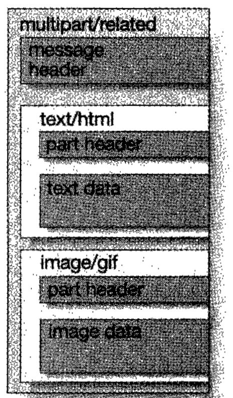
Creating an object representation of the incoming note is the responsibility of the client. One of the parser's more important design points is that it must not make a copy of any arbitrarily large message fragment (such as an entire GIF [Graphics Interchange Format] image), and instead use a bounded amount of storage (by processing that GIF image a buffer at a time). This necessarily precludes the alternative design point of building an in-memory object structure holding the entire message and then returning that object structure to the client. The parser's design takes the client on a guided "tree walk," as parser and client traverse the message's abstract syntax tree together. This design allows the client to efficiently map from the MIME grammar to the client's own message-store structure, without making intermediate copies. This design choice also implies that the parser, while thread-safe, will itself be single-threaded; the parser maps from a linear input stream to a linear output call sequence.

The MIME generator. It is the responsibility of the generator to build and format an outgoing MIME message, given some header information and zero or more body/attachment streams. The generator handles such things as formatting all of the keyword/value pairs, folding any excessively long lines, transfer encoding all data, and (optionally) converting any text parts from the local code page to the most similar Internet character set.

The client provides the generator with an output stream object, which will eventually hold the outgoing message, and zero or more input stream objects, from which the generator will read the body of the message and its attachments. Additional "hooks" are required for the generator to obtain from the client the values of the MIME header fields (such as "To:"

THE MIME PARSER INTERFACE

MIME Message Structure



MIME Parser Calls

```
BeginPart

  BeginHeader
  EndHeader

  BeginBody
       BeginPart
           BeginHeader
           EndHeader
           BeginBody
           EndBody
       EndPart
       BeginPart
           BeginHeader
           EndHeader
           BeginBody
           EndBody
       EndPart
    EndBody
EndPart
```

The client program interfaces with the MIME parser by passing it an input stream and subclassing certain "callback" methods. The MIME parser will parse the message in the input stream and will call these methods, allowing the client to control the parsing process. The callback methods fall into two groups: structural methods and header methods.

The *structural methods* are used by the parser to inform the client of the shape of the message, as given by the data-model grammar above. These methods are BeginPart, EndPart, BeginHeader, EndHeader, BeginBody, and EndBody.

The parser calls these methods as it processes the corresponding parts of the message, so that the client can track the current parsing state. They nest in an obvious way according to the message structure.

The *header methods* are called by the parser while a header or subheader is being processed. Their purpose is to inform the client of the tags, keywords, parameters, and values encountered in those headers. The client overrides only the header methods of interest to it. The header methods include Set_To, Set_Date, Set_Subject, and Set_Content_Type. (There are others, corresponding to other MIME header fields.) If a client is interested in handling a message's subject in a special way, it would override the Set_Subject method. When the MIME parser encounters the message's subject line, it will call Set_Subject, passing the entire field value. All header methods get the field value; other methods get additional information appropriate to the type of header field being parsed. For instance, Set_Date is given the date and time converted to the local machine's time zone, and Set_Content_Type is given the content type and subtype values as separate parameters.

Some methods, such as Set_To, are called repeatedly (once per recipient); other methods, like Set_Content_Type, are called exactly once. Some methods will not be called at all, if there is no header record to trigger them.

or "Subject:") in the various parts of the message, and other hooks obtain the recursive structure of the note.

In general, the generator's API is the inverse of the parser's API. Where the parser offers to the client all information that was gleaned from the input stream, the generator polls the client for the corresponding information, which it then formats and writes to the output stream object. Where the parser takes the client on a tree walk over the structure of the incoming note, the client guides the generator over the structure of the outgoing note.

Client considerations. To build new Internet clients or to enable legacy clients, the messaging objects and the MIME engine are coupled with client-side protocol objects. (See Figure 6.) For legacy clients that use a proprietary API (Microsoft's MAPI [Messaging API], VIM [Vendor Independent Messaging], and X/Open** API are examples of such proprietary APIs), the framework must be specialized to map between the object-oriented paradigm and functionality of the framework and the procedural paradigm and (usually less flexible) functionality of the API. These API-mapping subclasses of the framework typically operate at a loss of information, especially structural information, since the Internet e-mail model is structurally much richer than that of most proprietary systems.

The protocol objects can be put into two distinct functionality classes: one for accessing a message store or mail server, the other for submitting messages to a mail transfer agent (MTA). The messaging objects are used by the protocol objects to manipulate and store a message. The MIME engine is used by the protocol objects to convert between a MIME stream and the messaging objects as needed.

The interface for accessing a message store is defined by the Protocol class. This interface allows the retrieval of messages, either as a whole (POP3) or in parts (IMAP4), and provides folder (mailbox) operations. There are two implementations of this interface; one for the POP3 protocol, the other for the IMAP4 protocol.

The ProtocolSend class defines the interface for submitting a message to an MTA. This interface allows a connection to be established with the mail transfer agent and one or more messages to be submitted on the connection. There is an implementation of this interface for SMTP with extensions (ESMTP) where appropriate for clients.

To allow API-based legacy clients access to the Internet, one can build a framework specialization that maps the API to the framework objects. This is done by mapping the API calls to the appropriate messaging objects and protocol-engine methods. We have built such a framework specialization for MAPI, Microsoft's messaging subsystem in Windows 95**.

Due to mail-model restrictions in the APIs, particular restrictions may have to be enforced on the message store. If so, classes for the messaging objects must be subclassed for that particular message store. This was the case for MAPI, which has a mail model that is incompatible with the Internet mail model. New clients, or clients intending to become native Internet e-mail clients, will typically use the frameworks directly, rather than going through the usually lossy API layer. This allows the clients complete access to all information in the messaging and protocol objects.

Server considerations. As with client implementations, server implementations will have different issues depending upon whether they interface with new or existing message stores. (See Figure 7.) In either case, classes for the messaging objects must be subclassed to implement the access methods for the specific storage system (the file system or a database, usually). Existing message stores often present problems with data storage. Messages may be stored in a manner that makes it difficult or impossible to store some information required by the standard protocol. The message store may make retrieval of certain information more expensive than expected. The implementer may have to be very clever in order to get around some of the limitations imposed. The framework makes this job much easier than it would otherwise be; by centralizing these concerns in the message store classes, the implementer has a clean, canonical interface, common to all protocols, and need do the mapping only once.

We have implemented IMAP4 and POP3 servers on top of an existing proprietary mail server, based on earlier work, as a research project to validate the viability of the server frameworks. That implementation ran into many of the kinds of problems described above. In some cases a single protocol was used to transfer information between the proprietary client and the server and to transfer the same information from the server's memory to the message store. We

Figure 6 Client-side architecture

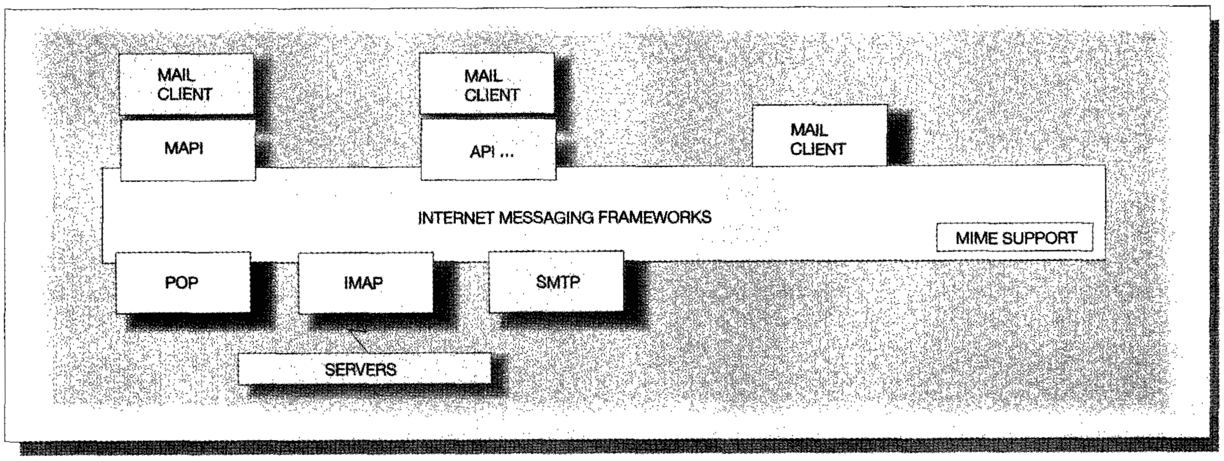
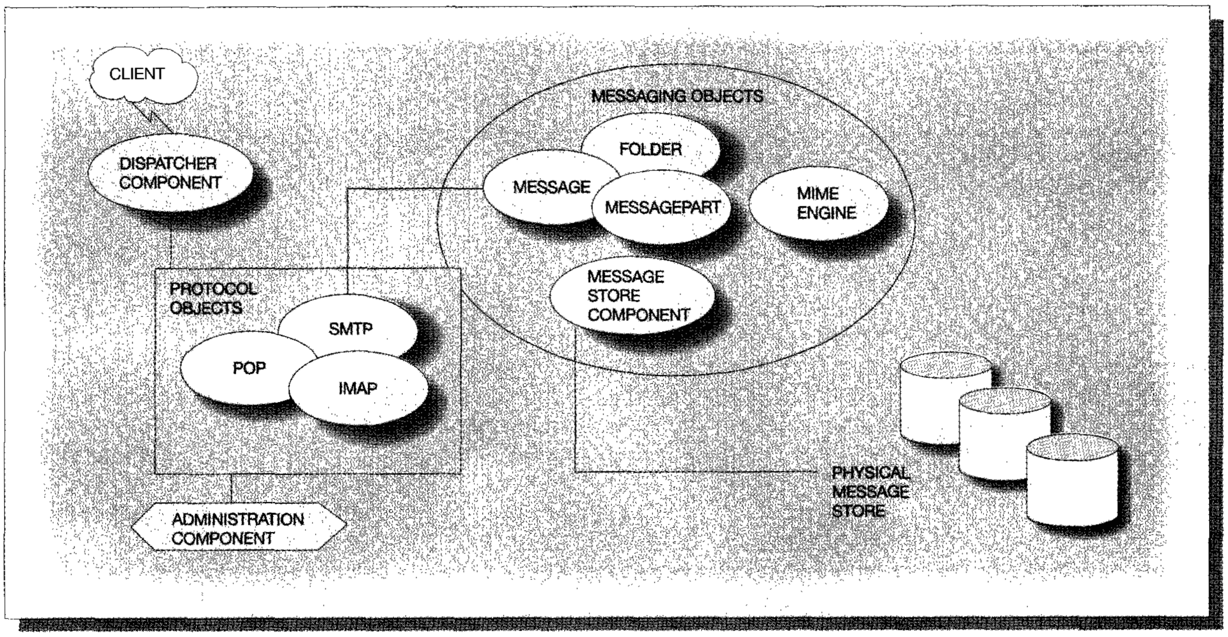


Figure 7 Server-side architecture



found, therefore, that we were not able to change the way that information was stored because the change would break the proprietary clients. In order to support both IMAP4 clients and proprietary clients on the same server, we had to make choices and trade-offs, and we had to sacrifice efficiency in some operations. Nevertheless, we were able to implement it quickly because of the flexibility provided by the framework, and once the backend trade-offs were made for IMAP, the POP implementation was a trivial extension.

Server implementations also involve two components not considered on the client side: administration and dispatching. In addition to mail handling, a server must authenticate users and allow administrators to do various things with the system. The protocols do not always provide the mechanism for server administration, so much of this is left to the implementation. For instance, an IMAP4 server administrator must be able to define users and create default mailboxes for those users, and this is completely outside the protocol (and the framework). Once the users are defined and a user tries to log on, the framework must allow the implementation to authenticate the user. This is done through the framework's administration component, which the implementation specializes to access the user-identification database created during the user-definition stage. For the IMAP4 protocol, the administration component also handles mailbox subscription operations.

A server is used by many users at once, and the connections from these users to the server require some management. With the POP3 protocol each user may have exactly one connection to the server at a time, and these are short-lived connections. (With POP3, one typically connects, logs in, downloads and deletes mail, logs out, and disconnects.) But with IMAP4, one client may have many server connections and these connections may persist for a long time. A client that allows a user to view several mailboxes at once, for instance, will have one connection per mailbox, and these connections may remain open and active for days at a time. The framework provides a dispatcher component to handle the management of these connections. Rather than dedicating one thread to each port, the dispatcher will listen for activity on a set of ports. The implementation subclasses the dispatcher's abstract class to handle data coming in on a port. The standard implementation will assign a "worker" thread from a pool of such threads, and will queue the request if there are no available threads in the work pool.

The dispatcher component feeds information from the clients into the protocol component, which analyzes the request. In IMAP4 and POP3, each transmission from the client begins with a command. The protocol component looks at the command, turns the request into one or more calls into the administration component or into the messaging objects (or rejects the request directly, as with an improperly formed command), and passes the work on to those components, which ultimately return data to the protocol component (from memory, from the message store, or from the administration process). The protocol component then packages that information, as defined by the protocol, and sends it back to the client.

Making the dispatcher a separate, distinct component was an innovation that evolved over time. Initial versions of the server framework portions had dispatching as an internal core function. This approach, while conventional, was not at all useful in helping to convert existing servers to IMAP or POP, since the existing servers already had their own dispatch mechanisms. A novel approach was needed, where the dispatcher is almost external to the framework and can be specialized to take advantage of the existing dispatching system in a given server.

Conclusions

The Internet Messaging Frameworks are the distilled results of six years' experience in building elegant, reusable, and highly efficient Internet e-mail technology components. These frameworks, especially the MIME engine, incorporate not only the strict standards as defined in the RFCs, but also a fair amount of error-correcting behavior to cope with the realities of ill-behaved mail agents on the Internet.

The early implementation of IMAP4 clients and servers as research projects has led to a better understanding of the problems associated with incorporating this complex protocol into IBM's e-mail products. By learning "where the rocks are," we are able to guide the product development groups, sharing our knowledge and sharing our experiences, to produce better, more reliable product-level clients and servers.

All of the Internet Messaging Frameworks for clients were used to build the "Lotus Mail 4.5" mail client. This is a special version of cc:Mail's MAPI-based R8 client, which operates as a standard Internet POP3 client with all the power of cc:Mail's feature set.

The Java version of this framework is being used to build the mail components of Lotus's Java-based eSuite component architecture. (For more information regarding eSuite Workplace see http://www.esuite.lotus.com.)

Since the intent for the Internet Messaging Frameworks is modularity, other groups have used them selectively—just the MIME parser, to boost their MIME parsing capabilities, for example—to write SMTP gateways and POP3 and IMAP4 servers. These

INTERNET VERSUS PROPRIETARY MESSAGING

Which has the advantage? While Internet-standards-based systems have the advantage of global interoperability, the proprietary systems are leaders in functionality and to some degree in reliability and security. Most efforts are now concentrated on coming up with improvements to Internet standards and operations to get systems that satisfy global connectivity as well as secure and reliable functionality. (Advantage: ✓)

	INTERNET	PROPRIETARY
Mail format and integrity	Standardized formats. Attachments arrive intact. ✓	Plethora of formats. Attachments usually must be converted by a gateway, often with loss of integrity.
Mail transport	Standardized store-and-forward mail transport. ✓	Various, not necessarily robust, routing schemes and formats. Need gateways (often with loss of data) to route mail outside domain.
Gateways	None needed. ✓	Typically multiple gateways needed to connect various vendors. All with different behavior and loss of data pattern.
Client/server	Full client/server architecture for mail access. ✓	Often file-sharing architecture. If client/server, then either totally proprietary with unpublished interfaces or with published APIs. Typically protocols are not published.
Functionality	Current standards push interoperability and simplicity, not features. New standards are under development to add features such as security, calendaring and scheduling, receipts, transactions, etc.	Typically offer a much richer feature set; specifically, features needed in the business community, such as group scheduling support, receipt notification, guaranteed delivery, authentication and privacy, etc. ✓
Administration tools	Typically done by editing configuration files. Tools are just now starting to appear.	Typically offer nice graphic tools to configure and administer the servers. ✓
Scalability	The Internet was designed from the beginning to be large and scalable. ✓	Varies widely among the vendors, but generally not very scalable.
Cost	Generally requires less server and administration support. ✓	Typically requires a lot of server and administration support, but varies widely among products.
Reliability/ security	Typically no control over routing and delivery of messages. Often not secure when transmitted in the clear. This is one of the key areas the IETF is focusing on improving.	Can provide notification and guaranteed delivery. Relatively high level of security within one network. ✓
Choice of clients	Any POP3 or IMAP4 client will work. ✓	Some offer choices, most will only work with the one proprietary client provided.

decisions were often based on the success of our MIME engine in handling all the MIME test cases in the MailConnect 1 interoperability test event organized by the Internet Mail Consortium, including the particularly difficult job of splitting and reassembling partial messages.

Other subsets of the frameworks are under consideration by many groups for use in converting proprietary mail systems to Internet-standards-based ones. The messaging objects, especially, are of interest as a good foundation for native Internet-mail object handling.

At this point the Internet e-mail community is very active in driving the standardization of many missing features: authentication, encryption, receipts, directory access, and others. We are participating in the standards development and are tracking and integrating these emerging technologies into the frameworks as part of our ongoing research and participation in these areas.

**Trademark or registered trademark of cc:Mail, Inc., Lotus Development Corporation, Sun Microsystems, Inc., X/Open Co., Ltd., or Microsoft Corporation.

Bibliography

Internet messaging is rapidly changing. Printed information, if available, is usually outdated. As with all Internet standards information, the only up-to-date sources are available on the Internet itself. Following is a list of Web sites that are major information sources for topics related to Internet e-mail standards.

All requests for comments (RFCs) and drafts are published by the IETF. Access to these can be found from the main IETF site http://www.ietf.org/.

The Internet Mail Consortium (IMC) maintains a thematically ordered list of e-mail-related RFCs and drafts at their site—http://www.imc.org/. This is the best site to start with for Internet e-mail-related information.

For information about IMAP a good starting point is: http://www.imap.org/.

Information about Microsoft's Messaging API (MAPI) can be found at http://www.microsoft.com/ win32dev/mapi/.

The X.400 specification was published by the CCITT (Comite Consultatif International Telegraphique et Telephonique) in 1992 in *Data Communication Networks: Message Handling System X.400*. Information about the X.400 publications can be found at the International Telecommunication Union (ITU) Web site http://www.itu.ch/.

For information about S/MIME a good starting point is: http://www.rsa.com/smime/.

For information about PGP good starting points are: http://www.pgp.com/ and http://bs.mit.edu:8001/~jis/pgp.html.

Accepted for publication September 5, 1997.

Jürg von Känel *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: jvk@watson.ibm.com).* Dr. Von Känel is the manager of the Multimedia Messaging group, which he helped found in 1991, at the Watson Research Center. Before that, he worked in the IBM Zürich Research Laboratory, where he worked on graphical user interfaces for an X.400 e-mail system, as well as a graphical editor for protocol flow-diagram specifications. He holds a master's degree in mathematics and a Ph.D. degree in computer science from the Swiss Federal Institute of Technology in Zürich (ETHZ).

John S. Givler *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: givler@watson.ibm.com).* Dr. Givler has worked in the Multimedia Messaging group at the Watson Research Center since 1994 and is the chief architect for the Internet Messaging Frameworks MIME engine. Previously he worked on the formal semantics of programs and the theory of term rewriting systems. He holds a bachelor's degree in mathematics from the University of Illinois, Urbana-Champaign, and a Ph.D. in computer science from the State University of New York at Stony Brook.

Barry Leiba *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: leiba@watson.ibm.com).* Mr. Leiba has worked in the Multimedia Messaging group at the Watson Research Center since 1991 as the chief architect for the e-mail server work. He has worked on the VM/ESA® (Virtual Machine/Enterprise Systems Architecture), VM/XA™ (Extended Architecture), and VM/370 operating systems since joining IBM in 1977. Mr. Leiba holds a B.S. degree in mathematics from the University of Florida and a master of science degree in computer science from George Washington University, Washington, DC.

Wolfgang Segmuller *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: werewolf@watson.ibm.com).* Mr. Segmuller has worked in the Multimedia Messaging group at the Watson Research Center since 1994. He is currently the chief architect for the e-mail client work. Since joining IBM in 1981, he has worked on mainframe system management and network management. He holds a B.S. degree in computer science and chemistry from Rensselaer Polytechnic Institute, Troy, New York.

Reprint Order No. G321-5660.