Facemail: Preventing Common Errors when Composing Email

by

Eric Lieberman

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2006

Author .....
Department of Electrical Engineering and Computer Science
May 18, 2006

Certified by .....
Robert C. Miller
Assistant Professor
Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Facemail: Preventing Common Errors when Composing Email

by

Eric Lieberman

Submitted to the Department of Electrical Engineering and Computer Science
on May 18, 2006, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Facemail is a system designed to investigate and prevent common errors that users make while composing emails. Users often accidentally send email to incorrect recipients by mistyping an email address, accidentally clicking "Reply-to-all" rather than just "Reply", or using the wrong email address altogether. Facemail is a user interface addition to an email client that provides the user with more information about the recipients of their email by showing their actual faces. This form of information is much more usable than the simple text in current displays, and it allows the user to determine whether his email is going to the correct people with only a glance.

This thesis discusses the justification for this system, as well as the challenges that arose in making it work. In particular, it discusses how to acquire images of users based on their email address, and how to interact with lists, both in learning their members as well as displaying them to the user. This thesis discusses how Facemail fits into current research as well as how its ideas could be expanded into further research.

Thesis Supervisor: Robert C. Miller
Title: Assistant Professor

Acknowledgments

Thank you to everyone who helped me to put this thesis together. I'd especially like to thank the following people in no particular order:

- My family - for always being supportive and helpful in all aspects of this thesis and my MIT career.
- UID Group - for great feedback and pilot testing my user study.
- Greg Little - for his help on integrating face detection.
- Jaime Teevan - for discussing list arrangement strategies with me.
- My study participants - for participating in my study.
- LaVerdes - for being open 24 hours a day.
- The Coca-Cola Company - for creating Cherry Coke.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Email is pervasive throughout both business and social settings, and as a result, it has received a wide amount of attention from researchers. The problem of spam alone has spawned an entire field dedicated to filtering and prevention, while there is another industry dedicated to beating those filters. Email security has also received a great deal of attention, as researchers wish to prevent email spoofing and protect email confidentiality. Despite this attention, relative little time has been spent investigating less malicious forms of email problems. In particular, this thesis focuses on errors that users make accidentally when they are composing email.

There are three common kinds of errors that users make when composing email:

1. Incorrect recipients - the email is mistakenly sent to the wrong people.
2. Missing attachments - the email is sent without the correct attachment(s).
3. Improper forwarding - the email is sent to the correct people, but those people shouldn't have access to all of the data contained earlier in the email thread. Since email clients generally append all previous replies, it's easy to forget that sensitive information may be in the bottom of an email.

Several solutions already exist (see 2.5.2) that deal with the problem of missing attachments, and improper forwarding doesn't occur very often; therefore, the remainder of this thesis will focus on the problem of incorrect recipients.

1.1 Incorrect Recipients

Emails can be sent to incorrect recipients in a number of different ways:

- Reply vs Reply-to-all - If the user misclicks Reply-to-all rather than just the Reply button, he could easily accidentally send an email to an entire list of people rather than to the single intended recipient.

- Similar email address - If two different addresses have similar representations, then it's easy to confuse the two of them. For example, `ec@mit.edu` and `educational-counselors@mit.edu` seem similar, but go to very different lists of people. `ec@mit` is the list of all residents of the dorm East Campus at MIT, while `educational-counselors@mit` is a list of MIT alumni who interview prospective freshmen to see if they are a good fit for MIT. Both organizations often abbreviate themselves as the initials EC; hence, it is easy to confuse the two.

- Mistyped email address - If the user accidentally types an address that is almost but not quite correct, it may be difficult to detect the problem. For example, if the user meant to send something to `susanh@mit.edu` but accidentally typed in `susan@mit.edu`, it would be difficult from the text alone to realize the problem.

- Missing/extra recipient(s) - If the user was sending the email to a large list of people, it is easy to forget one or more addresses.

- Improper etiquette - If the user is unfamiliar with email etiquette, they might mistakenly send an email to many more people than they should. For example, a user trying to remove himself from a list might send an email to the entire list, thus spamming a large number of people with an irrelevant email.

Many of these situations are common, and preventing these erroneous emails would save users time and effort. As email is used for more and more sensitive information [1], it becomes more and more important that emails reach their correct recipients. In fact, the results from a mistakenly sent email can be particularly devastating. The

following section examines several real anecdotes of incorrectly sent emails with severe consequences.

1.1.1 Selected Erroneous Emails

Composition errors seem to be very common among email users, and when I casually asked people about these types of situations, most of them had a good story to tell. Here are a few of these stories:

- A TA for a class at MIT had just written an exam for the students of that class. He decided to send it via email to the professor so that he could check it. However, he accidentally sent it to the list for the entire class. Since the class now knew the entire exam, the test had to be postponed until the TA wrote an entirely new one.

- MIT's admission process involves alumni called "Educational Counselors" who interview prospective students to see whether they would fit into MIT. When the counselors have interviewed a student, they send a report on the student to an educational counselors department at MIT, and the admissions staff can then look through it. This office often abbreviates its name as simply "EC".

  At MIT, there is a dorm called East Campus. This dorm also goes by "EC" and the email address `ec@mit.edu` corresponds to the list of all residents of the dorm. About once or twice a year, this list receives highly personal information about specific freshmen from educational counselors who don't know the correct address for the Educational Counselors' office.

- In Raleigh, North Carolina a company's employee mistakenly clicked "Reply-to-all" rather than "Reply" and sent very personal medical information to the entire company [12]. Another employee was so embarrassed for her that she was quoted as saying "I couldn't even make eye contact with her."

- Sometimes, though infrequently, erroneous emails can cause technical problems. This story involves a large company with several thousand employees. The

company had a single email list which contained the email address of every person who worked there. One person mistakenly sent an email to this list instead of a much smaller one. Soon thereafter, many employees sent replies to this email to the entire list saying "take me off this list". Since this list was very large, and there were many users trying to send email to the entire system at once, the mail server failed and just shut down.

The abundance of stories such as these suggests that users often encounter these problems, and therefore a solution that prevents them could be very useful.

1.2 Our Approach

Users often don't pay enough attention to the recipients fields (to/cc/bcc), and as a result they can miss the information stored there. In the small amount of time when the user is focusing on the recipients field, the interface should provide him with as much information as possible. The actual faces of the recipients are a natural source of information because humans are particularly good at recognizing faces [7]. This choice also appeals to the idea that "a picture is worth a thousand words." Given the choice of using faces, several design principles influenced the way in which those faces should be incorporated into a system.

1.2.1 Design Principles

Firstly, the system should be primarily based around a user interface. A purely backend system might try to determine the proper recipients to an email automatically and then adjust them accordingly. However, this is a very hard problem because it amounts to understanding what the user's intentions are. Since email is used in so many different contexts [1], this would be a even more difficult. By contrast, a user interface solution would simply provide the user a better mechanism to confirm that his email is in fact going to the correct people.

Secondly, the system shouldn't change the interface or workflow of writing an

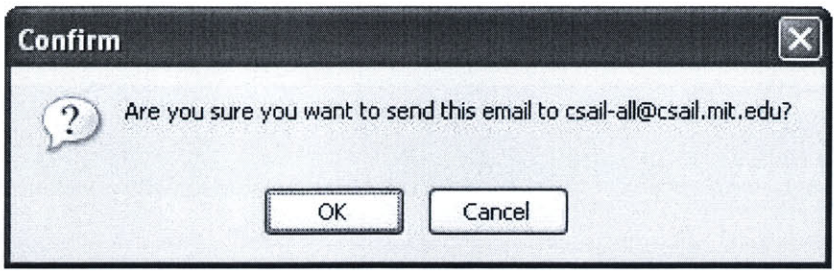

Figure 1-1:
An example of a user interface addition that would not succeed. If a user clicked on send and the above dialog showed up, it would not be effective in preventing errors because the user would simply ignore it or disable it.

email drastically. One of the nicest features of email is its ease of use, and adding extra steps or hurdles that a user would need to work through would remove that advantage. Even though there are a large number of email clients today, most of them use similar email composition interfaces; incorporating faces should not break this consistency.

Lastly, the system must not detract from the overall user experience of writing an email. Facemail must be both fun and useful if people are to be willing to use it. An annoying interface would probably be ignored or not used even if it provided some benefit. Therefore, a naive approach like popping up a dialog on clicking the send button such as in Figure 1-1 is doomed to fail.

1.2.2 Addition to the Compose Window

Following these design principles, Facemail simply displays the faces of the recipients in the mail composition window of an email client. The faces show up automatically, and they do not interfere with the user's primary task of composing an email. The faces do not require the user's direct attention to interpret; instead they are a *peripheral interface*, which is an interface informs the user without demanding his direct attention. Adding faces to the composition window also makes writing an email feel more personal and enjoyable, so it does not detract from the user experience. Figure 1-2 shows the current Facemail interface design.

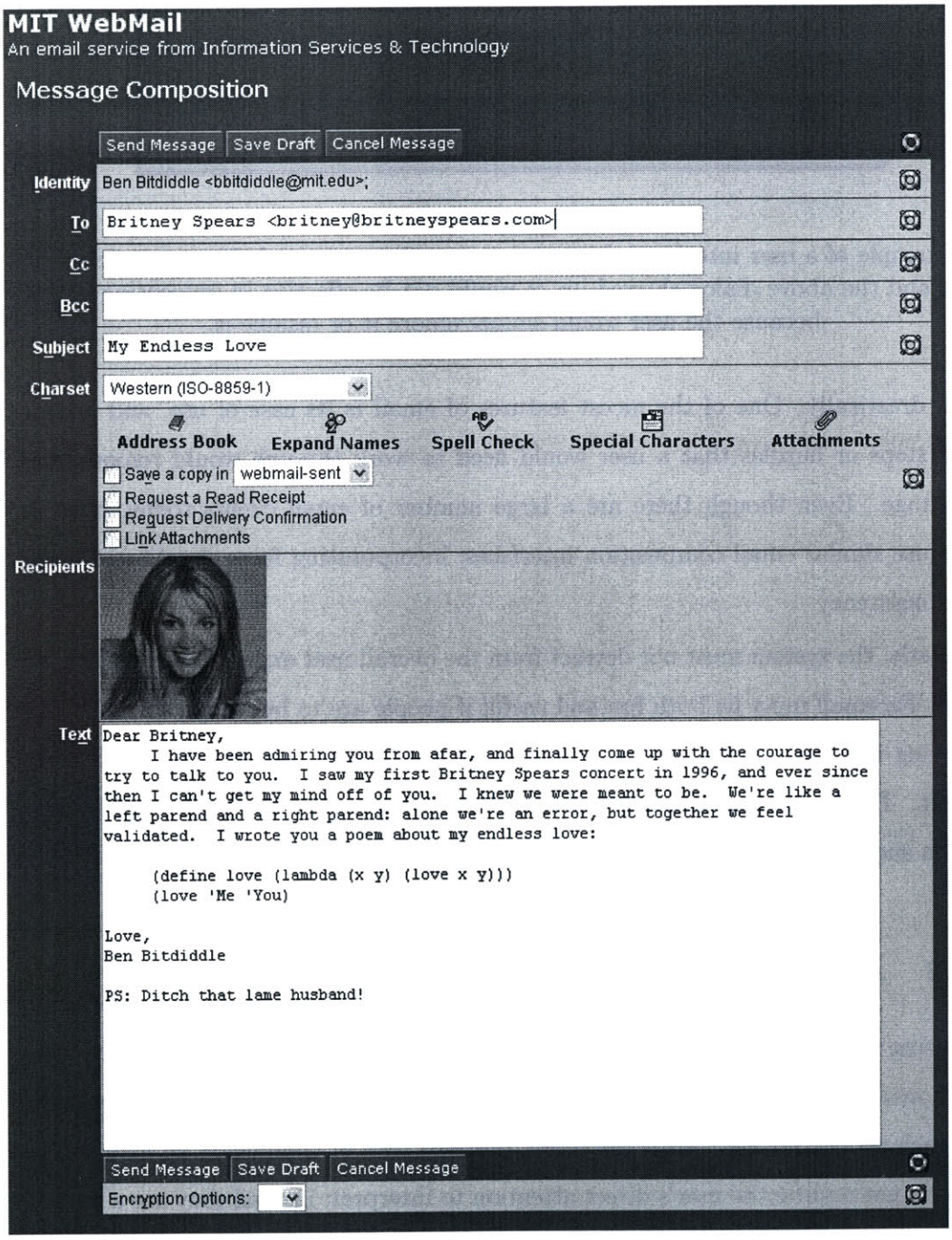

Figure 1-2: The current design of the Facemail system. The images of the faces corresponding to the recipients are shown above the composition text box.

1.3 Implementation

Facemail is currently implemented as a client-side system that runs entirely on the user's computer. The system is responsible for determining which images to show the user for given email addresses, as well as how to display them.

1.3.1 Backend

Facemail's backend runs in Java, and is responsible for three main tasks: determining list membership, finding images for email addresses, and rendering those images into a concise format for the screen.

Facemail tries to display faces for all of the recipients of an email, so if that email is going to a list, it needs to know all of the members of that list. To do this, the system tries several different list lookup mechanisms to find the members of the list. If the address is not found to be a list, it is assumed to be a simple single recipient.

Once Facemail knows the email addresses of the recipients, it needs to find a face for each of those recipients. Facemail checks many different sources such as Google Images and Facebook for images, and then attempts to select the best one.

Knowing the recipients and having images for all of them, the system then renders each list into a representation of its members. This representation contains the faces of the list in a way that allows the user to get a sense of the purpose, size, and members of the list.

1.3.2 Frontend

Facemail currently runs on MIT Webmail by using a Mozilla Firefox plugin called Chickenfoot [2]. Chickenfoot allows the local web browser to parse and modify live web pages using Javascript scripts.

Facemail's frontend takes the recipients directly from Webmail's to, cc, and bcc fields and passes them to the backend to find and create images for. When the backend has found the images, Facemail uses Chickenfoot to modify the Webmail compose window to include the images of the recipients' faces.

Because the information found by the backend may not be correct, Facemail also provides mechanisms to manually change all the information it collects. Users can modify address information such as the descriptive name on an email address or list membership information. They can change the images used for particular addresses, and modify the order in which images are displayed in lists.

Chapter 2

Related Work

Email composition errors have not received much research attention, although there is a great deal of research involving many related aspects of email usage, security, and software design. Several commercial applications have also developed features that are related to the goal of preventing erroneous emails. This chapter discusses each of these topics in turn, and shows how Facemail relates to them.

2.1 Email Usage

Email has become an extremely popular tool in both work and social contexts. In a work environment, email is used as a primary means of communication and information management [1], and the ramifications for changing any email system are widespread. Because of the relative flexibility of the original Internet standard [9], and the ease with which a user can write an email, it is very simple to accidentally send an email to incorrect recipients with no warning signals. Lists are resolved entirely server-side, and therefore the email client has little to no information about the list membership. Facemail helps prevent unintentional, non-malicious errors by gathering information not found in a normal email system.

2.2 Security

Secure and confidential information is often sent via email, and therefore email errors can be seen as a security breach. In this light, Facemail can be treated as a security system with many similarities to security research performed for email and other software systems.

Garfinkel et al. interviewed 470 Amazon.com merchants about their work with digitally signed email [6]. They found that although many users did not understand email security systems, they nonetheless were concerned about email security and wanted their emails to be signed. Because an incorrectly sent email can cause a security risk, it is likely that users who appreciate the importance of digitally signing emails would also like to insure that their emails reach the correct people.

Other studies [13, 5, 8] have shown that even though a software security system may work well, if the user interface isn't crafted carefully, the system becomes unusable. Facemail uses a simple, user interface based solution so that it can avoid the pitfalls of complicated metaphors and underlying mechanisms. Facemail's approach requires no extra learning of tools or security models, and therefore the user can avoid some of the problems presented by previous interfaces.

2.3 Interface Design

Reeder and Maxion show that in situations where the user's output doesn't necessarily match his goal, providing the user with better information can greatly increase their performance [11]. They showed how adding more information to the interface for setting up Windows security settings allowed users to drastically increase the rate at which they could modify them correctly. Facemail takes this approach by adding more information to the email client so that the user can make a more informed decision.

Facemail is a peripheral display that does not require the user to focus directly on the faces in our interface. It is intended to instill in the user an awareness of the

recipients of his email, much in the same way previous works have instilled awareness of current events or announcements [3]. Plaue et al. showed that peripheral displays with graphics convey more information than those with just text [10], and therefore faces are a good source of information.

2.4 Face Recognition

Biologists have long recognized that humans are particularly good at recognizing faces. There is debate over whether this is an innate ability or an acquired preference [7], however all agree that even newborn children can recognize faces very well. Because of this fact, faces are a very good source of the information, and representative icons such as AOL Buddy Icons would not have the same psychological impact that a face does.

2.5 Current Systems

Several current systems help prevent email composition errors using a variety of techniques. The next few sections describe some of these systems and how they relate to Facemail.

2.5.1 Auto-completion

Many different email clients offer auto-completion capabilities that help prevent errors when the user is typing an email address. Figure 2-1 shows several of the auto-complete features in different programs. Apple Mail's auto-complete, in particular, highlights known addresses at a specific domain in a different color. For example, if emailing within Apple, all known addresses ending in "@apple.com" are highlighted with a specific color to show that they are good addresses.

These systems help catch typing mistakes, but they don't solve the problems with "reply" versus "reply-to-all" or similar names that are both valid email addresses

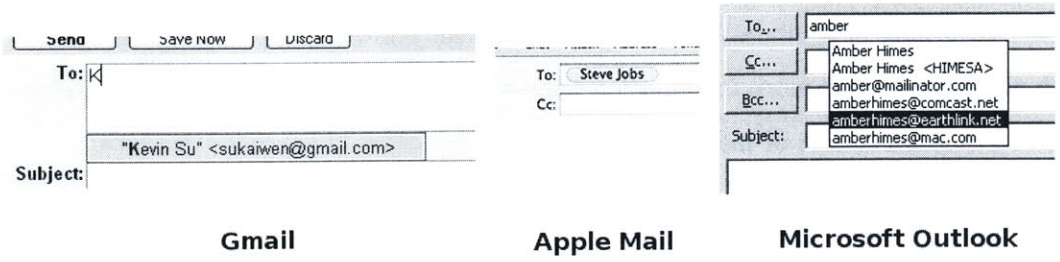

Figure 2-1: Several different auto-complete features.

Program Name	Email Client
SSW LookOut!	Microsoft Outlook
Attachment Scanner Plugin	Apple Mail
AttachmentRemember	Mozilla Thunderbird

Table 2.1: Several missing-attachments plugins.

("susan@mit.edu" vs. "susanh@mit.edu"). Auto-complete is a very nice feature, and it can easily be used along with Facemail to improve error prevention.

2.5.2 Missing Attachment Reminders

Many systems can currently detect missing attachments and prevent the user from accidentally sending emails without them. A cursory web search quickly comes up with several missing attachment plugins as shown in Table 2.1, and there are probably many more.

These addons function by scanning through the body of the email and looking for words that would signal that there should be an attachment (such as "see attached"). If the user presses send and their email has those keywords but no attachment, the user is prompted before they are allowed to send the email, as shown in Figure 2-2.

2.5.3 Gmail Pictures

Google has recently introduced a new feature called Gmail pictures, where each user can select a picture for themselves. Other Gmail users are able to see that user's picture, or pick a picture for that user themselves. Instead of automatically displaying

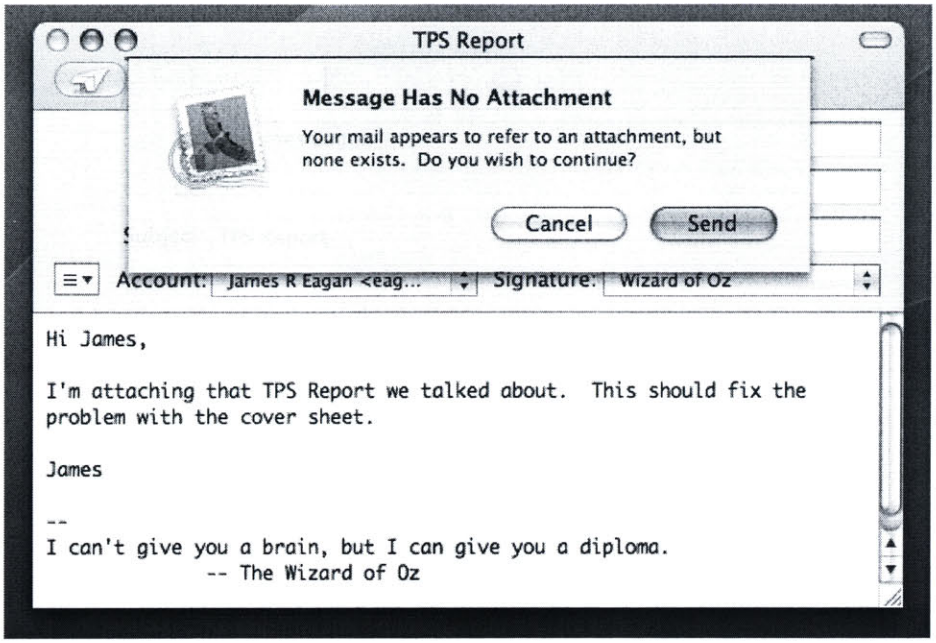

Figure 2-2:
A missing attachments reminder from the Attachment Scanner Plugin for Apple's Mail.app.

those images though, Gmail requires that the user mouseover an email address in order to see the picture. Figure 2-3 shows Google's concept behind Gmail pictures.

This system doesn't seem useful in preventing errors in the way Facemail does, but it could be a very good source of images for addresses at Gmail. By only showing the picture on a mouseover, the user doesn't verify the recipients of their email without conscious effort, which is out of place in the normal process of writing an email.

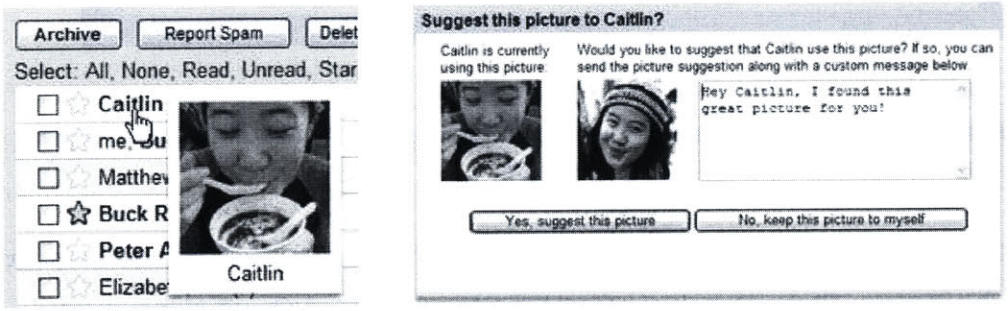

Figure 2-3:
Gmail pictures. Images are shown as mouseovers, and they are manually picked by each user and their contacts.

Chapter 3

Interface Design

Facemail integrates the individual's faces directly into the email composition window, rather than with a popup or mouseover. This approach does not require the user's direct attention; instead it acts as a *peripheral interface* which the user notices but does not have to focus on. Facemail does not add any extra screens between clicking "Compose" and actually composing a user's email, because that would detract from the user's ability to compose an email quickly.

3.1 Screen Location

All of the images in Facemail are located in an area below the message headers but above the compose textarea. In this location, as the user begins to write his email, he can't help but notice the pictures with a glance, even though he does not need to focus any direct attention on them.

If the email has recipients in more than one of the to, cc, and bcc fields, labels appear on the front of that line so that it is easier to distinguish which address corresponds to which picture. Figure 3-1 shows an example of where the faces fit into the current MIT Webmail interface. If there are multiple recipients of a single type (for example multiple "To" recipients), then they are simply stacked to the right as shown in Figure 3-2.

When the compose window is originally opened, the face area is minimized and

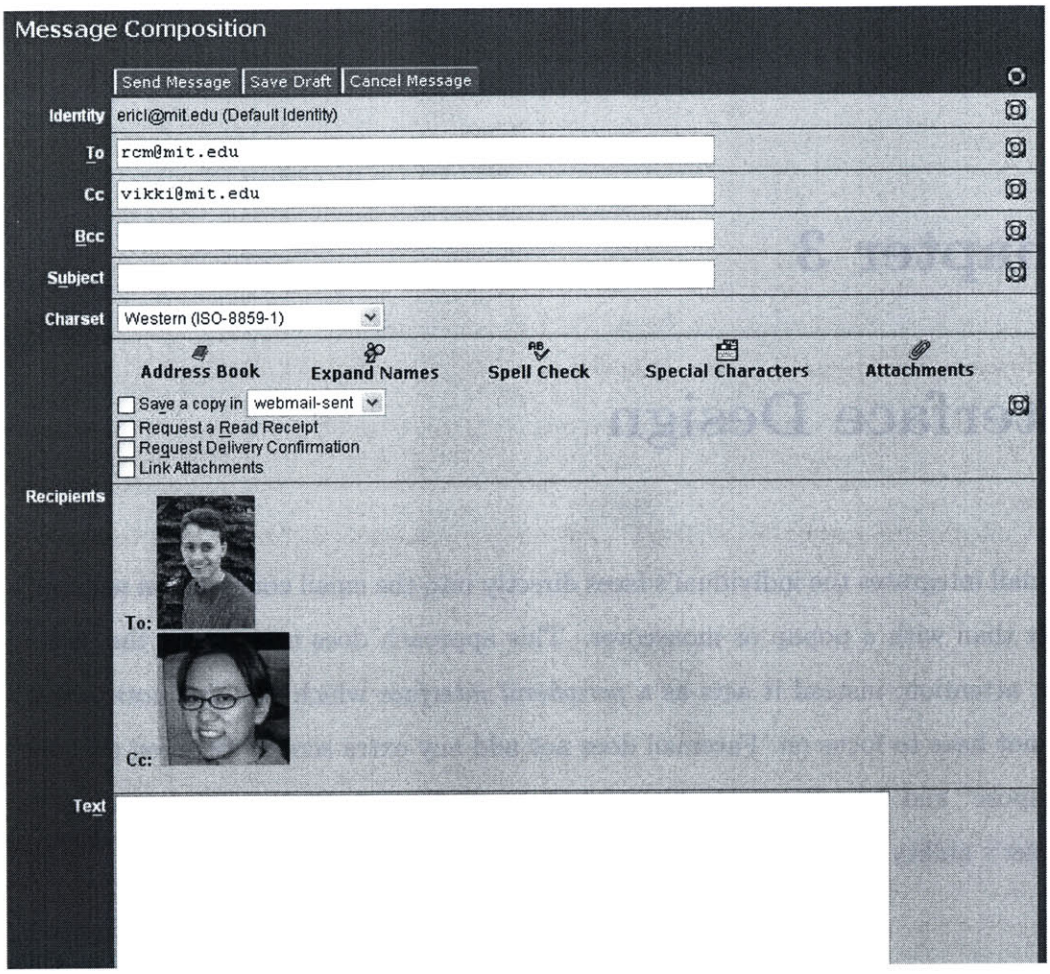

Figure 3-1:
The current interface's location in the compose window. To, Cc, and Bcc recipients each receive their own line of faces in order to preserve the spatial relationship of the textboxes above.

contains no images. When the user enters some recipients into the to, cc, or bcc field, the system automatically expands the face area and inserts the faces there. If the user reaches the compose window by clicking "reply" or "reply-to-all," then the faces are displayed automatically at the beginning. At all times when recipients are entered into the to, cc, or bcc fields their faces are shown; no user interaction is required to make them appear.

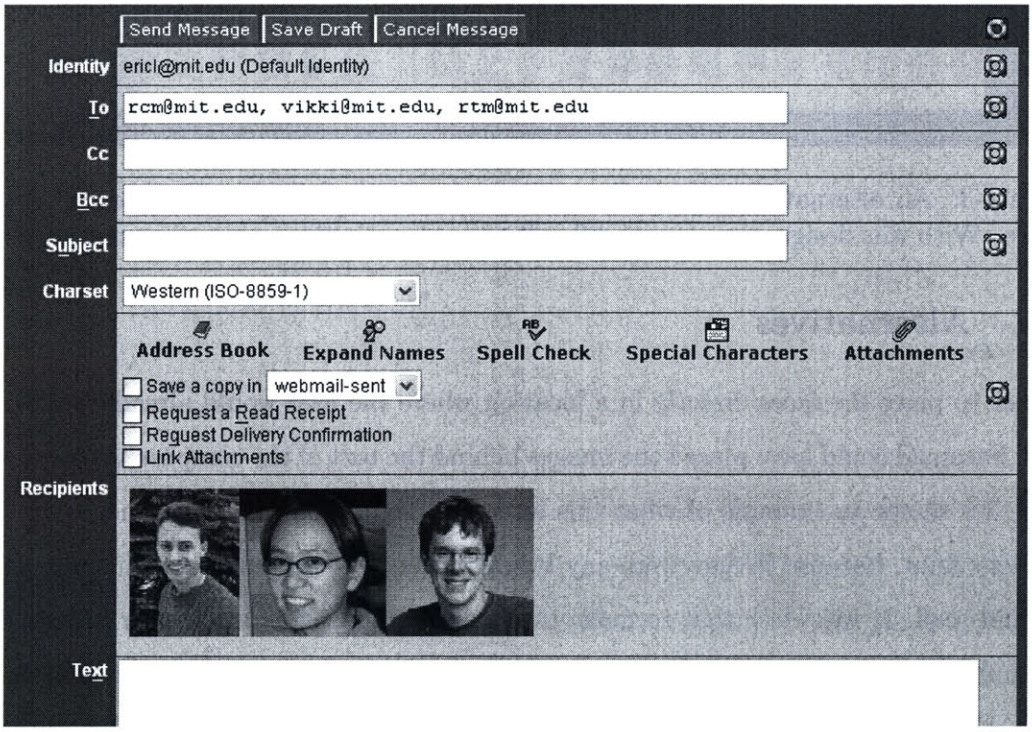

Figure 3-2: Faces for an email with three "to" recipients. The images show up in the order that they are typed in the above field.

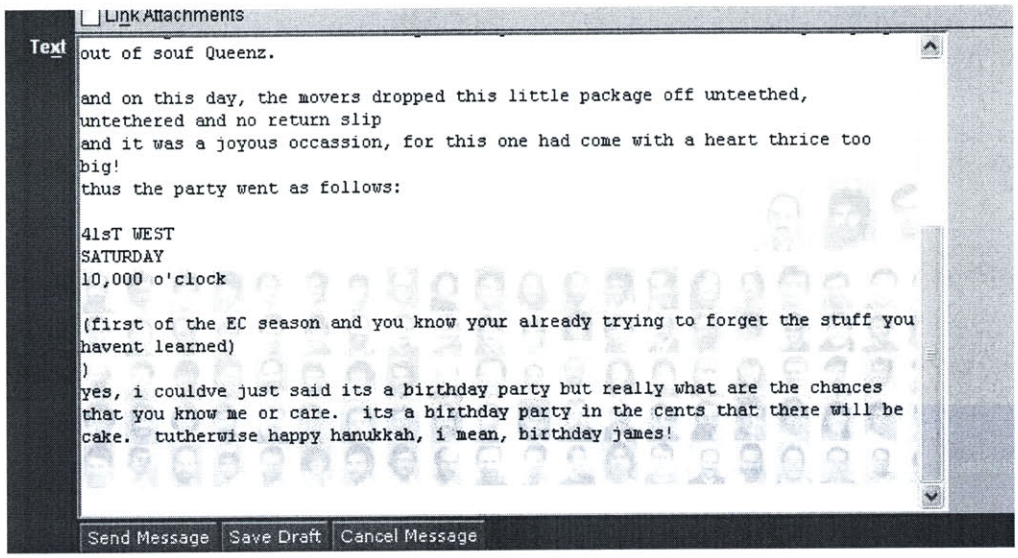

Figure 3-3: An alternative design where face images show up behind the compose textare. With this design, it was very difficult to read and write text in that textarea.

3.1.1 Alternatives

In order to place the faces directly in a location where the user would already be focused, Facemail could have placed the images behind the text of the compose textarea. Figure 3-3 shows an example of what this interface might look like. The images appear very faint, because if they were any brighter, the text became very difficult to type and read. It was clear that to make the images bright enough to really provide more information would detract from the user's primary task of composing an email.

Another approach would be to place the images directly next to the corresponding recipients field. Figure 3-4 shows an example of how this might look. Since the user's focus is on the compose textarea while writing his email, this location might be too far away and easily ignored. However, it does provide a nice mapping between the email address itself and the image. This interface warrants further exploration though, and it would be interesting to compare error rates between this and the current interface.

Because in email both the to and cc fields serve the same function, it might be beneficial to combine faces from the two fields into one display. In this situation, because Bcc has a different meaning, Facemail would need to use some method to separate bcc out from the other two. The faces for bcc recipients could be moved spa-

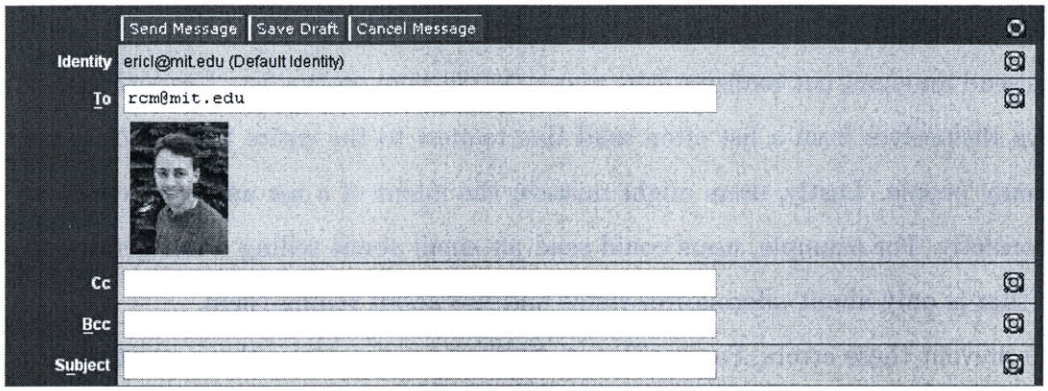

Figure 3-4:
An alternative design where face images show up directly under each corresponding recipients field. This design places a nice spatial mapping between the text and image, but might be easily overlooked because the user's focus of attention is too far away in the compose textarea.

tially away or dimmed out to show that those recipients are receiving less information. Future iterations of Facemail may investigate this alternative.

When the user has recipients in the to, cc, and bcc fields, the current Facemail display uses a great deal of screen real-estate. This large screen presence allows the user to easily see the images, however it is also obtrusive and may intrude too much on the user's space. Future versions of Facemail may investigate the tradeoff between how much space the display takes and how successful the interface is at error prevention.

3.2 Displaying Lists

Since email lists are common, it is important for Facemail to provide a good mechanism for displaying the membership of those lists. Because lists can potentially have thousands of members, this display must be concise but still representative of all of the members.

There are several errors that users make with incorrectly sending emails to lists. Firstly, they might send an email to a list that contains a person who they should not send the email to. For example, a surprise party invitation a group of friends

might accidentally include the friend whose surprise party it is for. Secondly, users might send an email not realizing how many people were on the list. Users trying to remove themselves from a list often send this request to the entire list, which is far too many people. Lastly, users might mistake the intent of a list and send email to it incorrectly. For example, users could send an email about selling a refrigerator to a list that is only about offering free items and not about selling them.

To prevent these errors, Facemail's list display tries to answer three questions:

1. Who is on the list?
2. How many people are on the list?
3. What is this list for?

If a list display correctly answers these questions, then users should have a much better idea about the membership, size, and purpose of the list, and catch errors before they are sent.

Many different representations of lists attempt to answer these questions. Some of these representations are shown in Figure 3-5. Each representation is of a list of around one hundred email addresses, and they are all created using the same face images.

(a) **Grid.** This approach works well for small to medium sized lists, however with large lists the faces become unrecognizable.

(b) **Crowd.** This approach conveys only a sense of the size of the list but not the actual people on the list. It would best be used as a component of another renderer which displayed those list images which were too small in this more representational form.

(c) **Scaling.** This approach tries to display several faces at a large size, while scaling the rest into a large crowd of people. This display shows a large amount of people while still showing clearly that they are people and not just dots. The following section explains how this effect works in more detail.

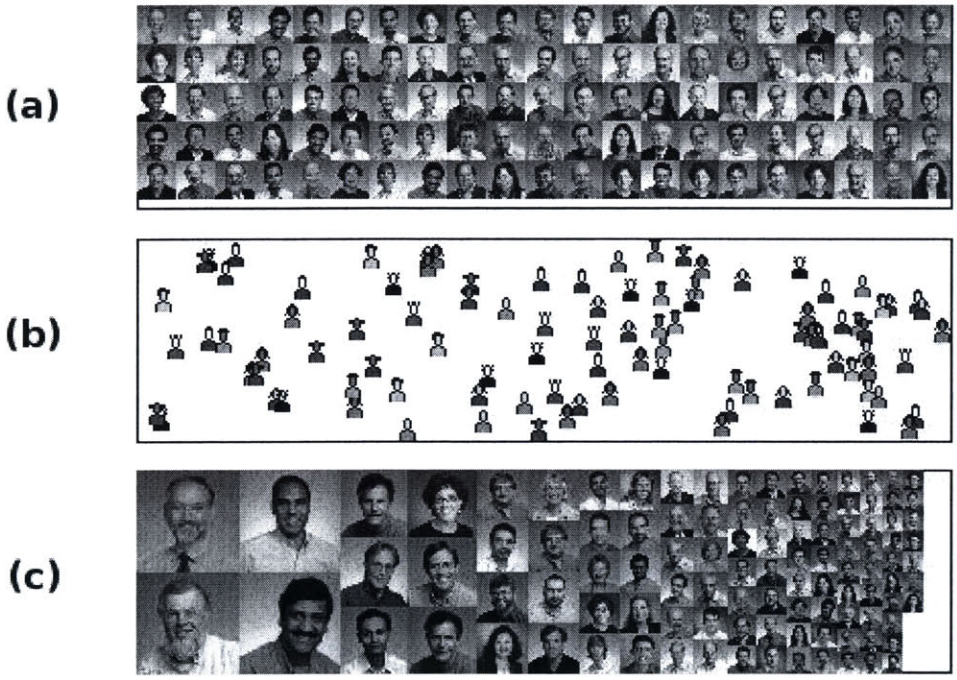

Figure 3-5:
Alternative ways of displaying lists. All of these are a representation of a list with around 100 members.

3.2.1 Scaling List Display

Scaling list displays are able to display a few images at a large size, and then use the rest of the images to give the user a sense of how many people are actually on the list. In a large list, a few people are probably more important to the user than the rest of the list, and the scaling list display tries to show those people at a larger size.

Currently, individual importance is determined by how good of an image the system can find for that person. In this way, the images shown at large scale are generally high quality faces, and even if the smaller images in the list are lower quality, the user still understands clearly that the list is made of people and not simply icons.

Given a rectangular area and a set of images, the scaling list display tries to fit all of the images at the largest size possible. The number of images in each column increases linearly, with a constant called the scaling factor. The scaling factor is determined by first guessing a very low value, and then increasing it until it reaches a value that is near optimal for filling the entire image rectangle. Appendix A shows the pseudocode for this algorithm.

3.2.2 List Image Enhancements

The scaling display gives a good indication about the number of people on the list as well as a display of the important people on that list at large size. It does not, however, give the user a good idea about the purpose of the list. By adding a few features to the scaling display, it is easy for Facemail to give the user much more information. Figure 3-6 shows several of these features.

(a) **Drop Shadow Text.** This approach adds text with a drop shadow over another renderer. The technique looks good against images with high frequency (such as having lots of people), however it was hard to read for images with fewer people.

(b) **Outline Text.** This approach adds outlined text on top of another renderer.

Figure 3-6:
Enhancements on top of list images. The information contained by the image can be enhanced by adding text or a logo about the list's purpose.

Figure 3-7: Three examples of list images at different sizes. List (a) has 10 members, (b) has 100 members, and (c) has 1000 members.

(c) **Logo.** This approach adds a logo to another renderer in order to give the user a quick way to see what a list is for.

Facemail uses approach (c) from Figure 3-6 as its list display because it is the best at displaying information that answers all three questions from Section 3.2. The few, large faces show the most important people on the list, while the crowd of small images gives the user an idea to how many people are on the list. Figure 3-7 shows how the approach scales to several different amounts of people. Finally, the logo provides a quick identifier for the purpose of the list.

3.3 Editing Interface

Because the images that Facemail finds automatically are not necessarily correct, it is important that the user has a mechanism for easily editing Facemail's automatic choices. To do this, the user can simply click on an image in the compose window, and

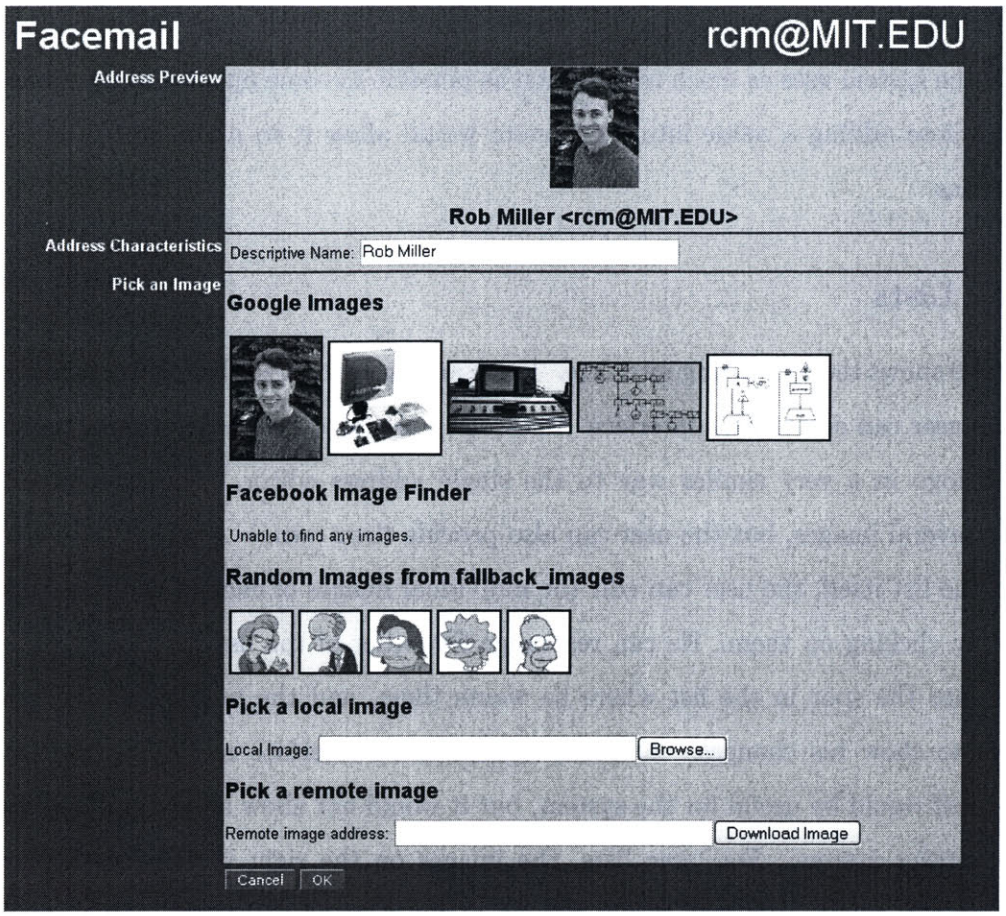

Figure 3-8:
Facemail's single address editing interface. Facemail uses several different image finding mechanisms to provide the user with image suggestions, and it also allows the user to supply their own images.

Facemail will pop up an editor for that address. The user can edit both individual addresses as well as lists.

3.3.1 Single Addresses

Figure 3-8 shows the editing interface for a single address. The user is presented with several choices of images that the system automatically finds, or they are allowed to supply their own. At the top, there is a preview of the image they have picked, and when they pick an image below the preview is dynamically updated.

Aside from the images itself, the user is allowed to enter or modify the descriptive

name for this email address (such as Eric Lieberman for ericl@mit.edu). When finding images, the system uses as much information as possible to come up with suggestions, and therefore adding a name into the system would allow it to make better image suggestions.

3.3.2 Lists

Figure 3-9 shows the list editing interface. There are many different list characteristics that the user can edit in this interface. First, they can modify the descriptive name and list logo in a very similar way to the single address editor. The system will suggest several images, but the user can also provide their own.

For the list itself, the user can edit the individual images of each of the members by simply clicking on them. He can rearrange the order of the list by dragging the images into the spot in the list where he wants them, and the preview at the top updates to show his changes. A direct manipulation interface modifying the list image itself would be useful for the system, but it would not allow all the capabilities of the current system. For large lists, the images on the right of the list display are extremely small, and they would not be easily selected or edited. If Facemail provided a zoomed in version to edit, the images on the left would be too large to be comprehensible. In the current display, every image is shown at a reasonable size and the user can edit each image easily.

A user can also edit the membership of the list by editing the textbox in the bottom of the display. This is particularly useful if the system is unable to find the membership for the list automatically; the user can simply copy the list membership into the textbox and then click "update list" to create a preview of the display of that list.

Because the textbox and membership display both show different representations of the same information, selection in done simultaneously between the two. For example, if the user is typing or selects text in the textbox below, that selection is reflected in the display above. If the user clicks on the display of an individual member in the top display, the corresponding text is highlighted in the textbox below. Figure

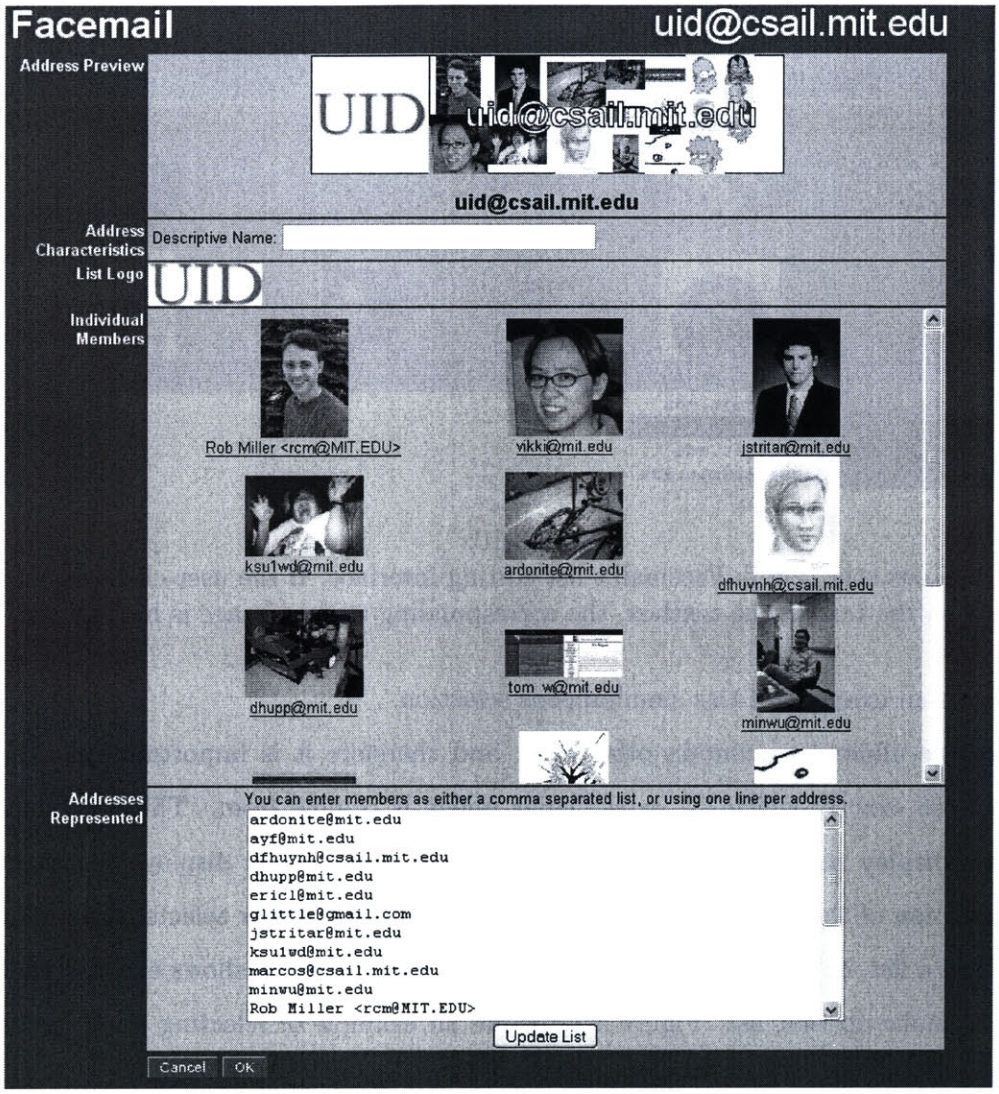

Figure 3-9: Facemail's list editing interface.

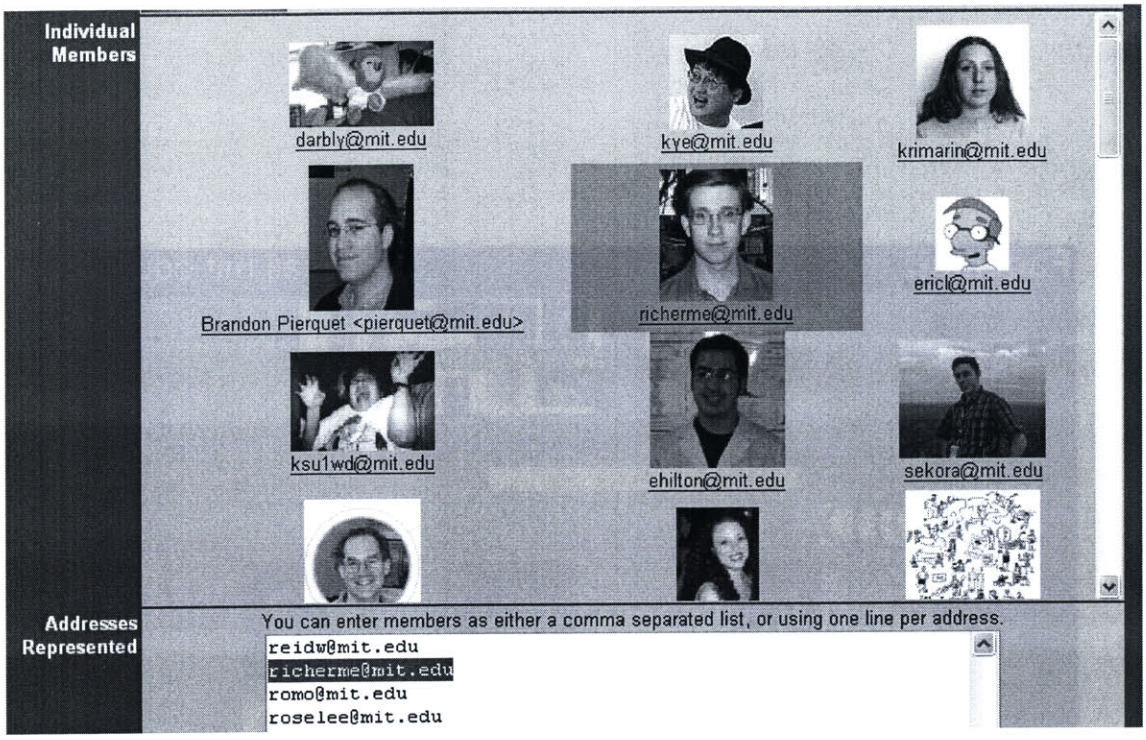

Figure 3-10:
Simultaneous selection in Facemail's list editing interface. If the user clicks on an image or on the text in the textbox, the corresponding text or image is highlighted.

3-10 shows an example of this simultaneous selection.

Lists are allowed to contain other lists, and therefore it is important that the user can see exactly which list a particular image is coming from. The individual members display shows only individual faces, while the textbox displays the exact representation of the list including sublists. Therefore, if the user selects a face that came from a list, or highlights a list in the textbox, the selection shows exactly what members make up that list. Figure 3-11 shows an example of selecting a list in the bottom textbox, and the corresponding highlighting above.

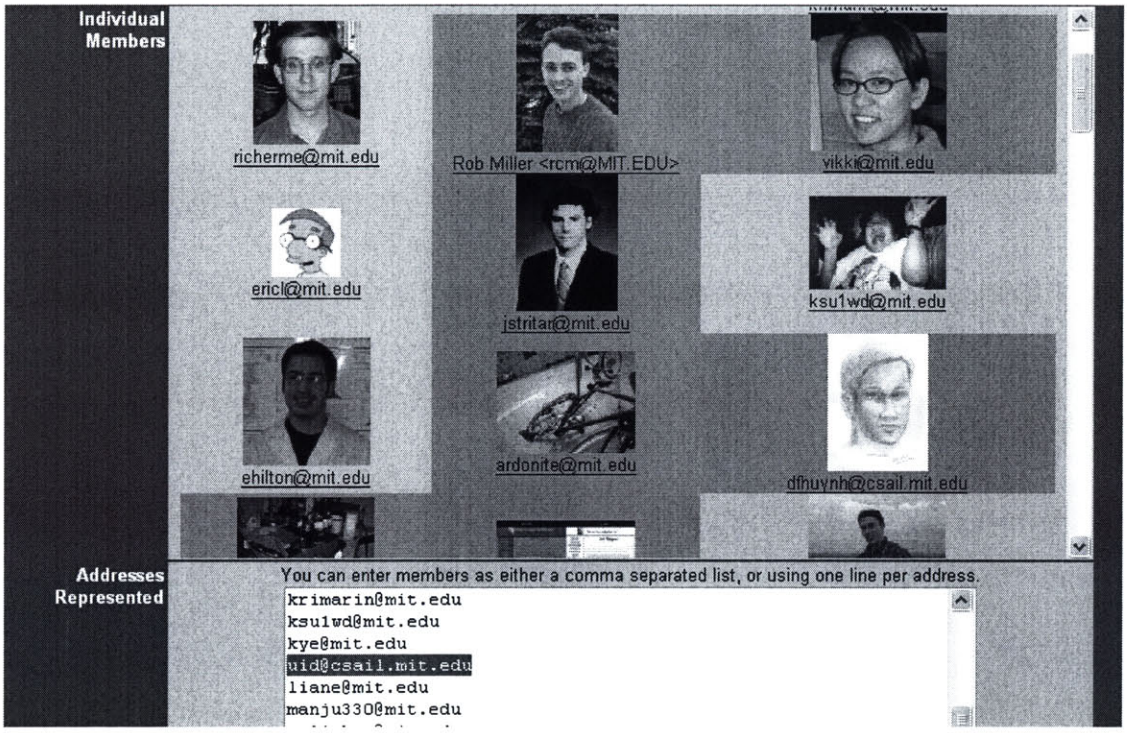

Figure 3-11:
List selection in Facemail's list editing interface. This list has a sublist "uid@csail.mit.edu" and because that sublist is selected in the textbox, all of its members are highlighted above.

Chapter 4

User Study

Before implementing Facemail, it was important to determine whether the concept would be useful. According to the principles behind Facemail, the error rate of a user with Facemail should be less than the error rate of a user without it. This hypothesis could be tested with a user study that mimicked users interactions with their email client both with and without Facemail. A user doesn't pay much attention to the to, cc, or bcc fields and therefore the user study could simulate that small amount of attention with a brief flash. After the flash, the study asked the user to answer questions based on the information that they were able to understand. The study hypothesized that a user would be able to answer our questions much more accurately using the faces as extra information. Because "a picture is worth a thousand words," even in the brief flash, the user could absorb a great deal of information from the picture that they couldn't from the text.

4.1 Experiment Setup

The user study was setup as a web-based test so that it could run a large number of subjects easily. The test consisted of two main parts: a questionnaire about the user's email usage, and a set of scenarios that the user should attempt.

After filling out the questionnaire, the user was shown the MIT Webmail interface in order to familiarize him with it (if he was not already). Then the test performed

many individual trials each consisting of three parts:

1. **Setup a Scenario.** The user was shown a scenario and asked to assume the role of an email sender. These scenarios contain short descriptions of the situation under which the user is writing this email, as well as the intended recipients and the body of the email. An example of such a scenario is shown in Figure 4-1, and all of the scenarios are shown in full in Appendix B..

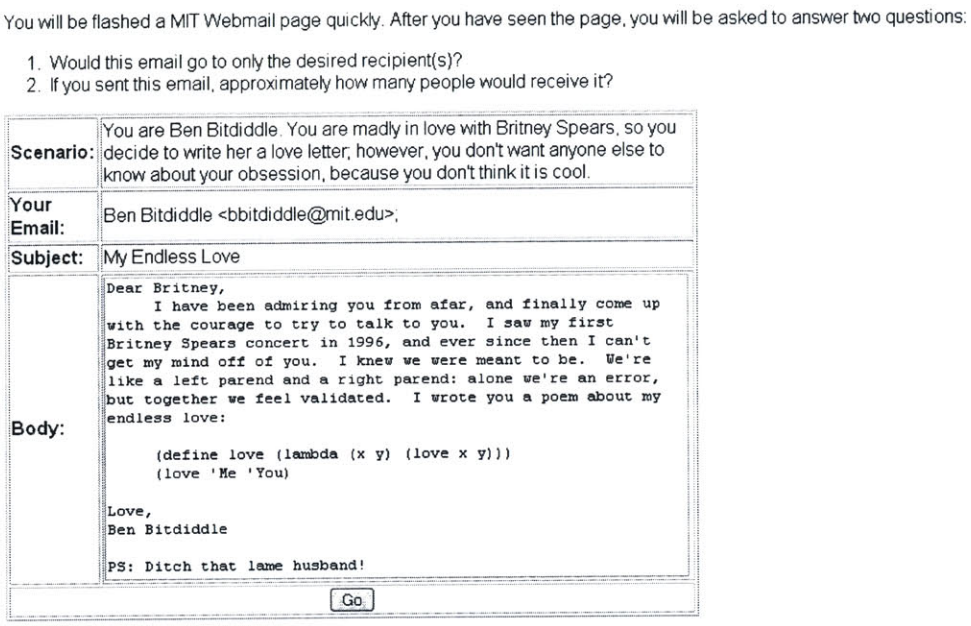

Figure 4-1: An example scenario from the pilot test.

2. **Flash the Interface.** Once the user understood the scenario, he was quickly flashed a screen of the Webmail interface either with or without faces. Some of the screens that were flashed had incorrect recipients, while some were correct.

3. **Answer Questions about the Interface.** After the flash, the user was asked two questions:

   - Would this email go to only the desired recipient(s)?
   - If you sent this email, approximately how many people would receive it?

The test recorded the user's answers to these questions, and then moved on to the next trial. For the first question, the user could pick between "Yes", "No", and "Not sure". For the second question, the user picked between "Less than 10", "More than 10", and "Not sure".

The test started with 3 warmup scenarios where the flash period was 10 seconds. These warmup scenarios were designed to get the user acquainted with the system, and with Webmail's user interface. The user was shown one scenario without faces, one with, and one with the wrong faces.

Following the warmup, the user study showed the user 30 different scenarios. The 30 scenarios were built using five different situations where a user might send an email. Each of those situations had three different possible sets of recipients, only one of which was correct. These three recipient sets along with the situation were shown to the user both with and without faces. Therefore, for each situation, there were six different combinations of recipients and faces.

The order that the scenarios were shown to each user was randomized, and the flash time got shorter as the test progressed: the first 10 scenarios had a 1 second flash time, the second 10 had .5 seconds, and the last 10 had .1 seconds. In each time interval, the test made sure to present 5 scenarios with faces and 5 without.

This study was submitted to and approved by MIT's Committee On the Use of Humans as Experimental Subjects (COUHES), with application #0511001472 for exempt status. Therefore, participants did not need to sign written consent in order to take part in the study.

4.2 Results

The web link to the user study was sent to two different email lists at MIT: free-money@mit.edu and ec-discuss@mit.edu. The first is a list for people who wish to perform in these kinds of studies, and the second is a dorm discussion list. The study was active for one week, and at the end of that week 84 users had completed the full test.

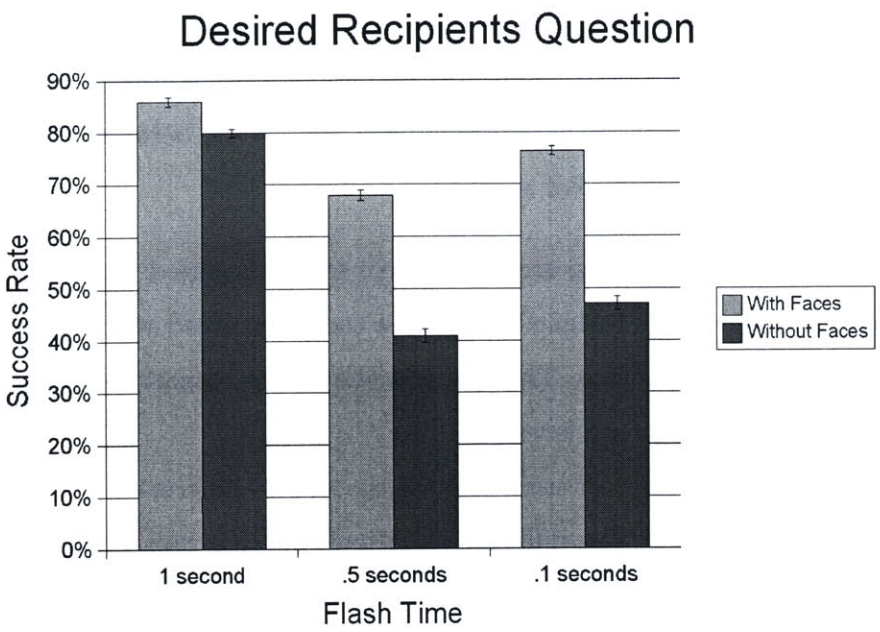

Figure 4-2: Success rates for the user study.

Sadly, due to a programming bug, most of the answers from the questionnaire were lost. Therefore, the following results are from the scenarios.

For each user we measured how many questions they answered correctly at each different flash speed both with and without faces. Figure 4-2 shows the average success rate users had at answering the question, "Would this email go to only the desired recipient(s)?" correctly. Any answer of "Not sure" was treated as incorrect in this graph.

For the question "If you sent this email, approximately how many people would receive it?" the answers came out very similarly. Figure 4-3 shows the average success rate users had at answering that question. Once again, any answer of "Not sure" was treated as incorrect in this graph.

An ANOVA statistical test determined that the average success rates between the flash times as well as with and without faces were statistically different. For the desired recipients question, the differences between faces and no faces were statistically significant with $p < 10^{-15}$ and the differences between flash times were also significant with $p < 10^{-19}$. For the second question the differences between faces and no faces

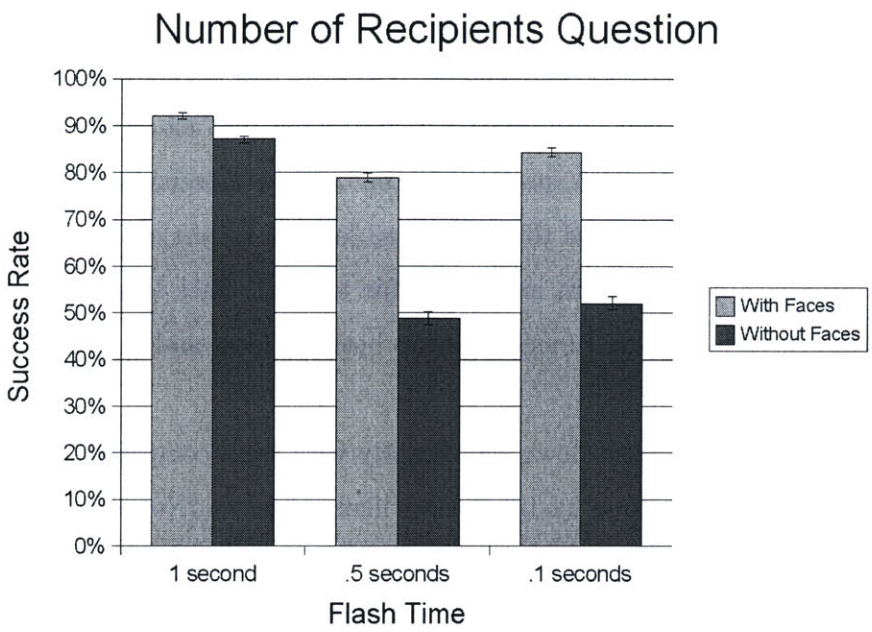

Figure 4-3: Success rates for the user study.

were statisticall significant with $p < 10^{-18}$ and the differences between flash times were also significant with $p < 10^{-17}$.

For the desired recipients question, 36% of the answers without faces (450 out of 1260) were marked as "Not sure," while with faces only 14% (181 out of 1260) were marked as "Not sure." For the number of recipients question, the results were similar with 35% and 13% (439 and 162 out of 1260) respectively.

4.3 Conclusions

The data in Figures 4-2 and 4-3 confirmed the initial hypothesis that the user would be able to gain much more information from the faces than just from the text. When the user was given enough time to fully absorb the information in the to, cc, or bcc field (1 second), there was not much difference between having faces and not having faces. However, when the user was not allowed to fully process those fields (.5 seconds and .1 seconds), the difference between having faces and not having faces was very significant. This result seemed to be empirical evidence that "a picture is worth a

thousand words."

An interesting observation of this data is that users did better with .1 seconds than .5 seconds. This fact can be explained by observing that all of the trials were done in order with decreasing flash speed: all of the trials at .1 seconds occurred after the trials at .5 seconds. Because of this, users had learned to process the data more quickly during the .5 second trials, and were able to apply that learning to the .1 second trials. More importantly though, the gap between faces and no faces remains essentially the same.

Also, users' uncertainty went down significantly with faces compared to without. Without faces, users were around 2.5 times more likely to select "Not sure" than they were with faces. Faces provide much more information, so users feel confident about making a decision with them as compared to without them.

Chapter 5

System Architecture

Facemail's architecture is highly modular, and allows for plugins in many different locations. This separation of responsibilities allows the system to be highly configurable and specialized to each individual user. Figure 5-1 shows an overview of the system architecture, and the following sections describe the design decisions that arrived at this arrangement of components.

5.1 Client vs. Server Side

Facemail could be implemented with a dedicated server for storing face images as well as list information. This server would be shared by all users, and they could upload their own photos to it. It would also need to be able to access different types of mailing lists, so that it could render list images accurately for lists maintained in different systems.

This server-based setup has several flaws. The first problem is one of bootstrapping: if each user is required to upload their own photo, then the system would not be very useful until many users had already uploaded their photos. Secondly, this approach brings up many privacy concerns. Users might feel uncomfortable sharing their picture with anyone who has access to the server, and they might wish for access control measures that would be difficult to administer. Along the same lines, a face server could determine exactly who each individual was emailing, and a malicious

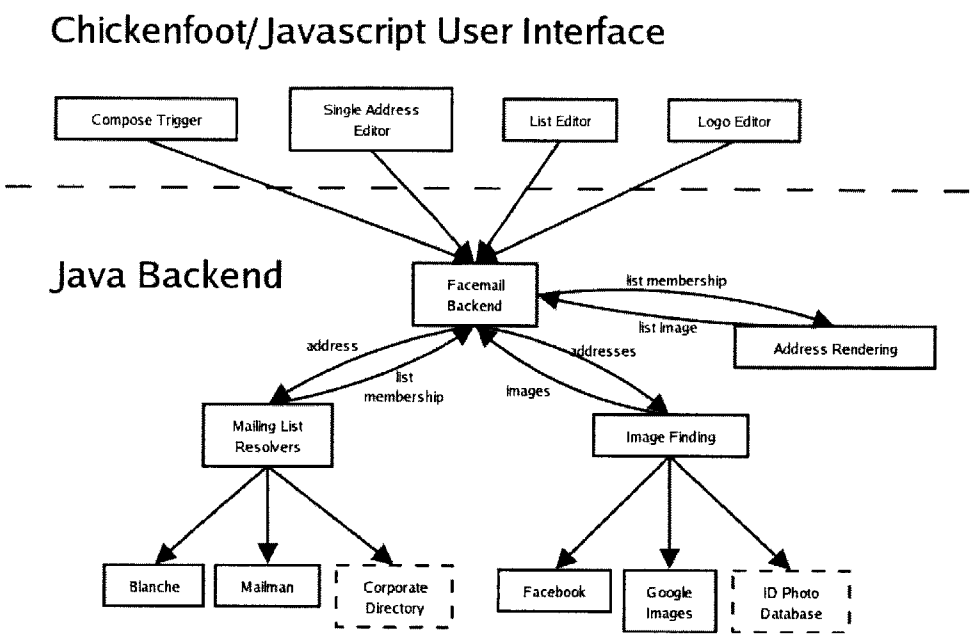

Figure 5-1: An overview of the Facemail architecture.

server could quickly use that information for spam or other purposes. Thirdly, getting list membership usually requires some form of authentication. This layer of security is required so that malicious users like spammers can not easily harvest email addresses, but it also prevents legitimate users from doing the same. Lastly, this approach is difficult to customize; users on their home accounts may wish to see different photos than users on their business accounts. For example, if a user is good friends with a co-worker, at home he may want to see a picture from a party that they attended, while at work he wants to see a more dignified picture. Because the database of images at home and at work would be different in this case, a user might be confused by the fact that a single email address could have two different images depending on his location. It would be interesting to determine whether this discrepancy would in fact cause errors.

Because of all of these problems, Facemail is implemented as a purely client-based system. The system stores all of its images and list information locally, and finds the photos using publically accessible methods. There are few privacy concerns, because Facemail does not add new data to the Internet; it instead only searches through data

that already is available. Since Facemail runs in the context of a person's computer, it has access to all of the authentication methods that that person does, which allows the system to do searches on password protected databases both for list information as well as photos.

The client-based approach can also be extended with the server-based approach. A voluntary server could provide information for people that wish to share their photo with the world (in fact, this is how Gmail pictures works), but that is not the only way to get a photo. If a user does not wish to share his image, it is possible for other users to still pick an image for him and use that on their local machines.

5.2 Choice of Email Client

Following the design principles from section 1.2.1, Facemail is integrated into a currently used mail program rather than being a complete mail program itself. Although the system might be more flexible in user interface design if it were a complete mail client, it also might not generalize well to a real email client. Writing a new email client would also incur large development costs from rewriting code that is already written in many current email clients. Given that Facemail should be a plugin in a current email client, there were several different applications that could support it:

- Microsoft Outlook Plugin - This is a widely used email program, and it has a plugin API that would probably use C++ or C#.

- Mozilla Thunderbird Extension - This is a free email client from the Mozilla foundation, and it has a full featured extension API that uses XUL.

- MIT Webmail with Chickenfoot - This system is generated entirely in HTML, which makes it easy to edit using a Firefox plugin called Chickenfoot [2] which allows users to modify live web pages using Javascript. The system also allows flexibility to quickly and easily expand to other web-based email clients, such as Gmail, Yahoo Mail, and Hotmail.

Facemail is currently implemented using MIT Webmail and Chickenfoot. The Chickenfoot scripts hook into a Java backend which could hopefully be reused for other email client plugins.

5.3 Address Resolving

One of the main tasks that Facemail has to accomplish is to determine whether an address represents a list of multiple people, or is just a single person. If the address is a list, then Facemail needs to find the members of that list as well. There is no single location that allows Facemail to get list membership for every list, and therefore address resolving is implemented as a plugin based system. All address resolvers must implement the following interface for converting a string address into an email address object.

```
public interface AddressResolver {
   public EmailAddress getAddress(String address);
}
```

The returned address may be either a list, a single address, or `null` if the address resolver cannot resolve a particular address. The plugin address resolver is a composite address resolver itself, and it queries all of the resolvers it contains and returns the first resolved address it finds. If no list address is found, the address is assumed to be a single person, and that address is returned.

5.3.1 Athena/Blanche

Lists at MIT are often managed using a system called blanche. Blanche is a command line utility accessible on Athena that will expand a list name into all of the users it represents. So, for Facemail, there is a system that automatically uses Kerberized SSH to log into Athena, and then query for list information. This list information is sent back to Facemail and then used to find images for the user.

5.3.2 Mailman

Mailman is another list maintenance system, and many different servers around MIT use it. Unlike blanche, where one access into Athena gives you access to a the membership of many different lists, each Mailman list requires a specific password for each list member. Mailman provides a web interface for retrieving list membership if the user knows the correct password for their account on the list.

Facemail currently stores a user's Mailman passwords in an local settings file that the user must configure. However, for a real deployed system, this is not acceptable because of the security implications of plaintext passwords. Also, because a user could be a member to a large number of Mailman lists, it is easy for him to forget or misplace his passwords for each list. If Facemail were deployed, it would be difficult for a user to find or remember these passwords, and therefore it is important that Facemail provides mechanisms for making this task easier. Since Facemail runs in the context of an email client, and Mailman sends a specifically formatted email telling the user of his password, it should be possible to automatically determine these passwords by scanning the user's email archive.

5.3.3 Others

Lists at MIT are generally managed using Mailman or Athena/Blanche; however, there are many other systems that manage list information. Facemail is set up in such a way that it can easily incorporate other such mechanisms. For example, in a corporate setting, a Microsoft Exchange or Lotus Notes server might manage the lists of the entire company. Each employee could query list membership directly from those servers. Facemail's plugin system is flexible enough that it should be extensible to other list management systems.

5.4 Image Finding

Finding an image of a person's face given their email address is a hard problem; many different websites or databases could contain these images, and some of those might be accessible only to authenticated users. Some people may not even have a suitable image accessible in any way to the system. Because of these problems, Facemail attempts to find a good default, but it allows the user to change an image whenever he wants.

Just as in address resolving, there are many sources of images, so Facemail uses a similar plugin based approach to find images. The main difference is that in address resolving, the first address that is found is assumed to be correct because each address resolving plugin is *precise*, meaning that if it has an answer the answer must be correct. Image finding, on the other hand is often not precise, and therefore Facemail provides a mechanism to choose between many different images. Each image finder implements the following interface:

```
public interface ImageFinder {
  public List<ImageRating> getImages(EmailAddress ea);
}
```

An image rating is simply the location of an image combined with a number between 0.0 and 1.0 representing how good the image is. For the default image Facemail simply picks the highest rated image. By allowing each image finder to determine its own rating, Facemail can represent the relationship that some image finders are trusted more than others, but still allow all image finders a chance to provide images. For example, a corporate ID database may return a rating of 1 for its image, while Google Images may impose a maximum of 0.5 for its results. If the corporate database contains an image, Facemail would use that, however Google Images could provide a default if the database was unavailable.

The image finder rating is also used to assign importance to newly resolved addresses. Because we want list images to show faces at large sizes, we bump those images that have good relevance ratings above those that have poor ratings. The

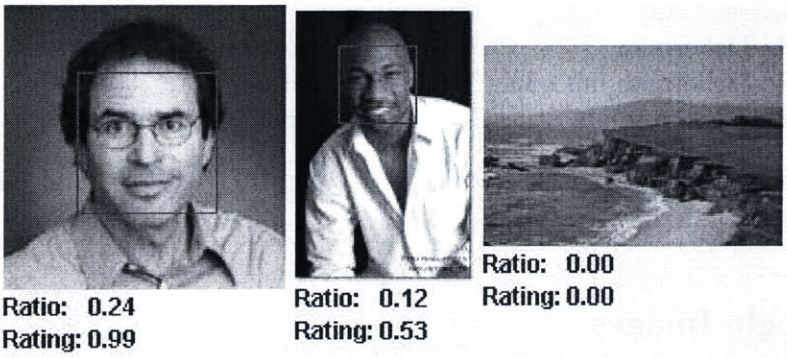

Figure 5-2: A set of example face detection ratings for various images, as well as the ratio of the size of the face to the size of the image.

final effect is that the right part of list images is made mostly of default icons, while all of the good results are shown on the left in a larger size.

5.4.1 Face Detection

For image finders whose source doesn't contain only faces, it is useful to rate resulting images using a face detector. In this way, an image finder can eliminate images that are not good portraits for Facemail to use.

The face detector calculates its ratings based on the relative area of the face in the picture compared to an ideal area ratio. The ideal ratio was calculated by taking a database of 92 portraits of CSAIL faculty and calculating the average ratio in those images. The average ratio of the face's area to the full image's area was 23%, and the Face detector rates images close to that ideal close to 1.0. If the image has a smaller ratio, then its rating decreases linearly to 0.0 as the area ratio decreases linearly to 0. If the face ratio is too high, then the rating also decreases, but only down to 0.5 for a ratio of 100% because an image that is purely a face is still a pretty good portrait for Facemail to use. Figure 5-2 shows an example of ratings that Facemail's face detector gives for various images.

ericl@mit.edu	ericl mit.edu
Eric Lieberman	Eric Lieberman <ericl@mit.edu>
Eric Lieberman mit.edu	Eric Lieberman mit
Lieberman mit.edu	Lieberman <ericl@mit.edu>

Table 5.1: Several different Google queries that could be tried for "Eric Lieberman <ericl@mit.edu>"

5.4.2 Google Images

The simplest image finder uses Google Images (images.google.com) to try to find images corresponding to an email address. No authentication is required, so Facemail can query the database directly. Also, the image thumbnails that Google returns are in fact just about the size of an image that we want, so we use those directly rather than downloading full sized images.

The search query for Google Images is currently very simple. For the email address "Eric Lieberman <ericl@mit.edu> the plugin queries for the email address, but replaces the '@' symbol with a space. So the corresponding query is "ericl mit.edu".

This approach because it was very simple, yet it seems to produce good results. Querying directly for "ericl@mit.edu" doesn't work well, because many websites do not directly contain email addresses in order to prevent spam. Also, querying for "Eric Lieberman" doesn't bring up the correct Eric Lieberman, since there are many people in the world with that name.

To enhance this plugin, one might consider trying several different searches and picking the best results from them. Table 5.1 shows some suggestions for searches that might be implemented. Most email addresses also use simple patterns for their usernames. For example, "ericl" comes from Eric Lieberman directly, and Facemail could use this fact to try to parse out real names from usernames. For example, a username "jsmith" could be parsed into the first initial "J" and then the last name "Smith." Using this information, even if the given address did not include descriptive information, Facemail could infer that information and perhaps get better results than it could without it.

5.4.3 Facebook

Facebook is a website (www.facebook.com) that is available only to college students at select universities. The premise of the site is to promote social networking and allow users to post pictures of themselves and their friends. As such, Facemail can use it to find faces, assuming that the user is also a Facebook member.

Facebook does not let users search directly by email address, although that address is stored in their database. As a result, Facemail only uses Facebook to query for full descriptive names, rather than simply the email address. For example, "Eric Lieberman <ericl@mit.edu>" would just query for "Eric Lieberman." Since Facebook localizes itself to the user's campus, results that were at the same campus would appear first. Since it is likely that a user's emails would be to people on their own campus, this rating helps us to choose between similar images. Also, since Facebook images are generally faces to begin with, the current plugin does not run the face detector on its image results. Instead, the plugin uses Facebook's search rank as the basis for the image rating.

5.4.4 Other Image Finders

Many more social networking sites like Friendster, MySpace, etc. could be queried for images in the same way Facemail uses Facebook. Using a combination of these services, it is likely that Facemail could find images for many people who have active pages on the internet. However, this is probably a small minority of email users, so Facemail offers the user the ability to add his own images quickly and easily.

Many companies have photo IDs and a centralized database with those pictures, so it would be ideal for Facemail to connect to that. Those image finders could simply return a rating of 1.0, since they are assumed automatically to be correct, and they would take precedence over any of the heuristic searches.

5.5 Caching

In order to speed up Facemail, as well as prevent load on the servers that Facemail searches, Facemail caches all of the images and list information it has found locally. This caching happens automatically during address resolving and image finding.

During address resolving, the plugin address resolver also contains an address database, which acts as both an address resolver and a place to store addresses. When the system asks the plugin address resolver for list membership, it first checks the cache and returns that if it exists. If it does not exist, it tries all of its internal address resolvers, and if any of them succeed it stores that success into its cache for future access.

In image finding, the main image creator which contains the set of image finders operates in much the same way. The image creator first checks the cache,and then if it does not exist checks all of the image finders it contains. When it has picked the best image, it stores that into an image database for future access.

Facemail currently does not refresh or update its cache at any times, but in a future version the cache could be set to update itself after a set period of time. Since list membership changes, it might be important for cached lists to expire quickly and force the server to refresh their membership frequently. For images, if a user places a good image of himself on the web, it would be useful to do a search again to find that image and update any default images currently being used.

5.6 User Interface

Facemail's user interface is implemented as a set of Chickenfoot triggers that modify MIT Webmail's live webpage. A Chickenfoot trigger is a Javascript file that is run whenever the url of a webpage matches a given expression. For example, the compose window trigger activates on the pattern "*webmail*compose.php*", and runs a script which inserts the faces into the compose window.

All of the Chickenfoot triggers connect to a Java backend using Firefox's Live-

Connect. Because the namespace for Chickenfoot is shared across all triggers, each individual trigger is wrapped in its own function which defines a local namespace. Currently, the Java backend is not shared throughout the system, however in a future revision it would be.

As shown in Figure 5-1, there are four main triggers which each provide a different component of the Facemail user interface. The first trigger is the compose trigger, which modifies the live Webmail page by inserting the faces into the correct location. Because finding images can be a time consuming process, the user interface is threaded, and the compose script constantly polls for updates to images as they are being created. For large lists especially, the process of finding images for all users on the list can take a long time, so during that time period it is important that the UI remain responsive.

Firefox's security model does not allow a remote webpage to include or link to local content, but in this situation, it is very important to be able to do that. For this reason, Chickenfoot has a method called `localUrl` that allows the system to securely bypass Firefox's safeguards. The system injects a local iframe into the compose window, and displays all of the images in that frame.

Beyond the compose trigger, the other three triggers are involved in editing the information stored by Facemail. The single address editor, list editor, and logo editor all trigger off of local files with a "file:///" beginning. In this way, these triggers can ignore the problem of Firefox's security model by simply running entirely locally. Each of these triggers operates in a popup window originating from either the compose window, or from the list editing window. When the popup is closed, the system needs to signal back to the original window that the user is finished editing. To do this, the original window inserts a function on the window object of the popup, and when the popup is closing it is required to call that function before closing entirely.

Chapter 6

Evaluation

Due to time constraints, Facemail has not been run through a comprehensive evaluation, but pecific components of the system have been evaluated separately when possible. I have also personally used the system, and have seen it prevent some of the errors it was designed to prevent.

6.1 Image Finder Performance

Two different lists of people with known faces were run through Facemail's image finders in order to test their effectiveness. The first list was of 63 current and former undergraduate residents of a particular dorm, and the second was the listing of CSAIL faculty. Both Facebook and Google Images were able to find some images for both lists, though neither had a particularly high success rate. The success rates for both plugins are shown in Table 6.1.

For different types of email addresses, the image finders have widely varying success rates. Facebook was roughly 27% accurate for students on the dorm list, but since faculty are very unlikely to subscribe to Facebook, the success rate for them was only 1%. Google Images, on the other hand, performed much better for CSAIL faculty who were more likely to have their own webpages than students. For faculty, our plugin's first choice was correct 45% of the time, while for students it was correct only 10% of the time. An interesting thing to note is that all of the faculty did have

		Google Images			
	List Size	First	Top 5	Wrong	None
Dorm List	63	6 (10%)	8 (10%)	1 (2%)	34 (54%)
Faculty List	93	42 (45%)	47 (51%)	11 (12%)	15 (16%)
		Facebook			
	List Size	First	Top 5	Wrong	None
Dorm List	63	17 (27%)	17 (27%)	10 (16%)	17 (27%)
Faculty List	93	1 (1%)	1 (1%)	27 (29%)	60 (64%)

Table 6.1:
Image finder performance. "First" is the number of results where the top ranked image was correct. "Top 5" is the number of results where a correct image was in the top 5 results. "Wrong" is the number of results where a face was picked, but it was the wrong face. "None" is the number of addresses for which no images could be found.

an image on the CSAIL website, but for some reason that webserver had blocked Google from indexing those images. Therefore, all of the results were from faculty members' individual web pages, although many of them used the image copied from the official CSAIL website.

Another interesting statistic to note is how accurate an image source is at finding the correct image and placing it first, not just in the Top 5. For Facebook, if it found a correct image at all, then that image was invariably first. On the other hand, Google Images sometimes placed the correct image farther down in the rankings. Using the face detector considerably increased the success rate of ranking images first, and it proved that it was useful for automatically finding good images.

This evaluation also showed us how valuable the face detection could be in some cases. For the faculty list with Google Images, six separate addresses had correct images where the face detection improved the rating such that the system automatically picked a better portrait than Google's top-ranked result. For the dorm list, this only happened one time, however given the low success rate of Google Images on the dorm list, this result is not surprising. Clearly, including face detection helped the system to find better results.

Different image finders also had different error rates for producing incorrect results. Google images rarely produced a face image that was not correct (12% on the faculty

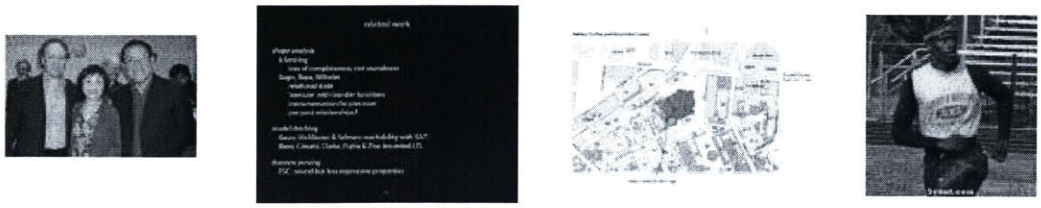

Figure 6-1:
Incorrect images found by an image finder. Image (a) contains the correct person, but he is standing with two other people. Image (b) is from a powerpoint presentation written by the person searched for. Image (c) is a map to a particular professor's location. Image (d) is a person with the same real name as the searched person.

list), but Facebook frequently chose incorrect images (29% of the time on the faculty list). Google images rarely didn't find any images for faculty, however many student addresses had no search results. Facebook was the opposite: for students it usually had at least one suggestion, while for faculty many addresses had no results.

Both Facebook and Google Images often fail to find good images for email addresses, but some of their failures offer insight into how Facemail could be improved. Figure 6-1 shows several examples of image finding failures that can be used to refine how Facemail finds images.

Sometimes, image finders find the correct person, but the photo contains extra people that make the image confusing. This problem suggests implementing automatic cropping, or picking images with single faces over those with multiple. Often, if the user has a webpage, there are graphics on that webpage such as a powerpoint presentation or an office map. The face detector helps to sort out this information and pick only good images. Lastly, since some names are very common, an image finder relying on the user's real name might find a completely different person sharing the same name. This insight means that Facemail should attempt to use email addresses (which are unique) before using real names in many cases.

6.2 User Experiences

Facemail has not undergone any comprehensive user testing, but from my own personal experiences I can say that Facemail is both useful at preventing errors as well as fun. I have accidentally typed "uid@mit.edu" rather than "uid@csail.mit.edu" and the fact that Facemail didn't display the correct graphic clued me in that I had typed the wrong address. It is also very interesting to look at and change individuals' pictures; sometimes the automatic suggestions are quite funny.

When first using the system, the faces were distracting when I was trying to compose an email. They were distracting for two reasons: 1) as the system downloads new images, the images are constantly changing and 2) I was not used to seeing the faces, so my attention was drawn to them. As I got more used to the system though, and as the system built up a database of images, I grew less distracted and was able to focus more on my task of writing an email.

Chapter 7

Conclusion

7.1 Contributions

In conclusion, Facemail's design and implementation allow it to provide a valuable service in preventing users from making several common errors when composing email. Not only does the system help prevent errors, but it does it in such a way that users will enjoy the system even as it helps them.

Facemail's user interface design is the product of many iterations of the design process, and it reflects a good balance between adding more information to the screen but not distracting the user too much. The user's innate face recognition skills provide him with extra information through a peripheral interface rather than a more focus-driven interface.

Facemail's system architecture is very powerful and flexible, and it allows customization and experimentation with many different mechanisms at all levels of the system. A future student or researcher could easily add extra plugins for address resolving, image finding, or address rendering. Also, since the entire system is client-side, users can adopt Facemail without needing special permissions or scripts on a server.

Facemail opens up many more avenues of research both in adding more capabilities, and in evaluating what has been completed. It would be very interesting to learn more about error rates both with and without Facemail, and it would be instructive

to be able to put a cost measurement to these errors. Facemail's libraries could also be used for a variety of new email techniques and capabilities involving security, error prevention, and general email usage.

7.2 Future Work

This thesis's contributions could be extended in many different directions by adding additional capabilities to Facemail, measuring error rates, or incorporating it into new settings. Facemail opens up several new avenues of research, and this section will discuss those avenues.

7.2.1 Error Rate Measurement

By simply asking users about their email experiences, it seems that email composition errors happen fairly regularly. However, it would be nice to have actual numbers and percentages to confirm our suspicions. A large scale user study could determine the actual error rates with which users make email composition errors.

This user study would ideally measure all the types of errors that users make, and how often they make them. What percentage of email generated is sent incorrectly, and are some types of emails more error-prone than others? Does the choice of email client affect the error rate, and does the experience of the user affect how often they make mistakes? How severe are the errors that users make? It would be interesting to try to assign a cost to each error in terms of money and/or man hours and then determine what is the most costly.

7.2.2 Measure Time Spent on To/Cc/Bcc

Users spend a very short amount of time looking at the to, cc, and bcc fields of their emails, and that is the cause of many incorrect recipient errors. A user study could test this hypothesis by measuring how much time they actually spend looking at those fields while performing real tasks.

This user study would need to examine several different scenarios, with several different email clients. In some situations, the user should type in the email address, while in others they should hit "reply" or "reply-to-all" and let the system do it for them. The system could artificially introduce errors, and see if the users caught on or not. Then, the study could compare the amount of time that users that caught the errors spent looking at the to, cc, and bcc fields versus the amount of time that users who missed the errors spent looking at those fields.

7.2.3 Improved Importance Ranking

The system is currently set up so that list images are drawn in order of importance of the user. Therefore, the most important people show up the largest and first in the list. However, the current mechanism for assigning importance simply relies on the automatic image relevance rating. It would be interesting to experiment with different methods to assign importance to individual users.

Since the system is running in the context of an email client, it would be easy to count the number of emails sent to and received from a particular user, and then assign an importance based on that. One could also give precedence to email addresses that the user had interacted with recently over those that they interacted with a long time ago. A more difficult suggestion would be to examine things such as corporate hierarchies and determine important people from that. For example, many companies have ways of looking up a user's supervisor, and that person should probably be considered very important.

7.2.4 Additional Capabilities

There are many places where a person could add capabilities to the current Facemail system. A user could implement more image finders and add them to the system. In particular, a corporate database or social networking site would probably be easy sources for more images. Also, since similar email addresses often represent the same person (`rcm@mit.edu` and `rcm@csail.mit.edu`), an image finder could search through

the current database of images and suggest from those that already exist.

Extra address resolvers could allow the user to view a larger number of lists correctly. One could write a plugin for Microsoft Exchange servers, or other proprietary list servers. Current plugins could also be enhanced to provide more information. For example, the Blanche plugin only returns addresses of the form "ericl@mit.edu", but it could easily use a tool like `finger` to determine that the full address should be "Eric Lieberman <ericl@mit.edu>".

One could also experiment with more address renderers in order to present the user with more information from the rendered image. One could try to highlight certain faces while dimming others, or perhaps add more text such as the number of recipients. These changes would have to be evaluated to determine how effective they are in preventing composition errors.

7.2.5 Business Setting

As has been mentioned several times throughout this thesis, implementing Facemail in a business setting would likely be more successful than the current implementation at MIT. In particular, the list management interface for the business could be leveraged to simplify address resolving, and a photo ID database could make image finding very simple. With those extra resources, the system would work very well, but it would be interesting to compare such a system to the current system at MIT. In particular, differences between the types of emails, types of users, and system setup would provide interesting information about the effectiveness of Facemail.

7.2.6 Email Security

Given Facemail's system which maps email addresses to images, it would be easy to implement a system that shows this image on incoming emails as well as in the composition interface. Since the from address of an email is trivial to spoof by a spammer, including the images would in some sense validate the email, and cause the user to trust the email when they shouldn't. Because Facemail is designed to prevent

errors, this feature would go against the principle behind Facemail.

For digitally signed emails, however, the recipient has already been verified, and it would not be a problem to use the face as a representation. Garfinkel et al. argue that email security is a difficult problem partially because there is no well-recognized user interface for representing a secure email [4]. Using the pictures of faces on signed incoming emails may provide an alternative to other user interface representations of security.

7.2.7 Image Correction

Images entering the system are often suboptimal for a number of reasons. If the image is not square, then it doesn't fit into the list displays as well and there ends up being gaps between images. If the image includes a face, but the face is very small relative to the image size, then the we would rather use just the face portion of that image.

Facemail could benefit from a set of tools to autocorrect images as well as allow users to modify images to make them fit to the system better. For example, using the face detector, Facemail could crop large images down to just the face stored within them. It could also crop rectangular images down to square images. If an image is not square but is a good portrait size nonetheless, Facemail might want to extend the background of the image in order to make it square but fit into lists better. These tools should be able to operate automatically to offer default suggestions, and they should also allow the user to modify images directly from within Facemail.

7.2.8 Shared Image Server

Facemail users would benefit from a custom server where they could upload their own pictures. Gmail pictures does this for Gmail users, and it would be nice to incorporate this feature directly into Facemail. In this way, rather than relying on heuristic searches, for many users we could rely on human-picked images that are probably higher quality. Other sources like Facebook could also allow a user to directly upload and manage photos directly from Facemail.

7.2.9 Large User Study

In order to evaluate its effectiveness at preventing errors, Facemail should be tested in both a field and lab study. For the lab study, an interface could intentionally change the "reply" or "reply-to-all" buttons to insert the wrong email addresses, and then determine whether the user notices the changes either with or without Facemail. After using Facemail, users would be able to comment on their perceptions of Facemail and whether they feel that it would be fun and/or useful.

For a field study, Facemail should be deployed to a large number of users and monitored to see how useful the system is to them. Users could submit examples of when Facemail saved them from making a mistake, and when it did not. For evaluation purposes, Facemail could add buttons to the email client that allow a user to signal to the system that it caught their mistake, and therefore the system could record statistics about how useful Facemail is to users. User comments would also provide a large amount of information about user perceptions of Facemail; is it a fun and useful experience, or does it detract from writing an email?

7.2.10 Email Enhancements

The information stored in Facemail could be used to implement several new email features.

- List Subtraction - If a user wished to send an email to all members of a list except for a few people, the user could easily do this with Facemail. Since Facemail knows list membership information, it can simply send the email to everyone except the specified person. This feature would be especially useful for things like surprise parties or gifts.

- List Password Management - Facemail needs to know list password information in order to correctly resolve list addresses. This information is often scattered throughout many different locations, and Facemail could provide a central repository and management system for list passwords.

Appendix A

Pseudocode for Scaling List Display

The following code describes the algorithm used to fit images into a rectangle for display. It assumes the variables `totalImageHeight` and `totalImageWidth` represent the rectangle's height and width respectively.

```
function CalculateScaleFactor(numImages) {
  guess = .1
  do {
    guess *= 1.05
    width = CalculateWidth(numImages, guess)
  } while (width > totalImageWidth)
  return guess
}

function CalculateWidth(numImages, scalingFactor) {
    columns = CalculateColumns(numImages, scalingFactor)
    imagesInColumn = 2
    width = 0
```

```
    for (i = 0; i < columns; ++i) {
        width += totalImageHeight/imagesInColumn
        imagesInColumn += scalingFactor
    }
    return width
}

function CalculateColumns(numImages, scalingFactor) {
    n = 0
    numColumns = 0
    inColumn = 2
    while (n < numImages) {
        n += inColumn
        inColumn += scalingFactor
        numColumns++
    }
    return numColumns
}

function DrawImages(images)
    imagesInColumn = 2
    scaleFactor = CalculateScaleFactor(images.size())
    x = 0
    for each image i in images {
        y = 0
        imageDimension = totalImageHeight/imagesInColumn
        for(j = 0; j < imagesInColumn; ++j) {
            drawImage(images[i], x, y)
            y += imageDimension
        }
```

```
            x += imageDimension
            imagesInColumn += scaleFactor
        }
    }
```

Appendix B

User Study Scenarios

The following scenario was used as the warmup test. Users first saw the correct recipients without faces, then with faces. Then they saw the incorrect recipients without faces. These trials were all flashed for 10 seconds to allow the user to become acquainted with the user interface.

Scenario	This is a warmup test. It is simply for the purposes of getting you acquainted with the system. You will have 10 seconds to see and examine the compose window. The email should be going to George Bush at the Whitehouse. Check the to address and any faces you may see closely to see if the email is going to the correct recipient.
User's Email	Joe Schmoe <joe@mit.edu>
Subject	Testing
Body	Testing! Thanks, Joe
Correct Recipient(s)	 George Bush <georgebush@whitehouse.gov>
Incorrect Recipient(s) 1	 Britney Spears <britney@britneyspears.com>

The following five scenarios were shown to the user in random order both with and without faces. Each scenario represents six different trials:

1. Correct recipient(s) without faces
2. Correct recipient(s) with faces
3. Incorrect recipient(s) 1 without faces
4. Incorrect recipient(s) 1 with faces
5. Incorrect recipient(s) 2 without faces
6. Incorrect recipient(s) 2 with faces

Scenario	You are Joe Schmoe, a concerned US citizen that would like to know more about the war in Iraq. You decide to go straight to the top and ask George Bush himself for some answers.
User's Email	Joe Schmoe <joe@mit.edu>
Subject	Concerned about Iraq
Body	Dear Mr. President, As an American citizen, I am very concerned with the state of the war in Iraq. I feel that it is in our best interest to leave the country as soon as possible and to bring our troops home. What is your current plan and timeframe for doing this? Sincerely, Joe Schmoe
Correct Recipient(s)	 George Bush <georgebush@whitehouse.gov>
Incorrect Recipient(s) 1	 George <george@whitehouse.com>
Incorrect Recipient(s) 2	   George Bush <georgebush@whitehouse.gov>, CSAIL <csail@mit.edu>

Scenario	You are a party organizer for csail. You've reserved East Campus's Talbot lounge for a party, and you want to invite everyone in csail to come.
User's Email	Party Chair <party@mit.edu>
Subject	PARTY TONIGHT!
Body	Clowns! Piniatas! Beverages! Oh my! It's the 5th First Annual Tuesday party! Come to East Campus Talbot lounge and dance dance dance the night away. Hit on that hot neighbor of yours! It'll be super fantastic! Starts at 10pm! -Party Committee
Correct Recipient(s)	 CSAIL <csail@mit.edu>
Incorrect Recipient(s) 1	 George Bush <georgebush@whitehouse.gov>
Incorrect Recipient(s) 2	 Britney Spears <britney@britneyspears.com>

Scenario	You are a dissatisfied MIT student who believes that student life on campus has been degrading. You decide to write to President Hockfield to indicate your displeasure.
User's Email	Ben Bitdiddle <bbitdiddle@mit.edu>
Subject	Student Life at MIT
Body	Dear President Hockfield, In my time as a student at MIT, I have watched many of the cultural traditions that I used to hold near and dear to my heart disintegrate. Rush was removed, and then all freshmen were forced to live on campus. There have many other signs recently that lead me to believe that the administration is trying ot turn us into an ivy league school, rather than the unique place that MIT is. What do you plan to do to preserve MIT's culture? Thanks, Ben Bitdiddle
Correct Recipient(s)	 Susan Hockfield <hockfield@mit.edu>
Incorrect Recipient(s) 1	 Susan <susanh@mit.edu>
Incorrect Recipient(s) 2	 CSAIL <csail@mit.edu>

Scenario	You are Ben Bitdiddle. You are madly in love with Britney Spears, so you decide to write her a love letter; however, you don't want anyone else to know about your obsession, because you don't think it is cool.
User's Email	Ben Bitdiddle <bbitdiddle@mit.edu>;
Subject	My Endless Love
Body	Dear Britney, I have been admiring you from afar, and finally come up with the courage to try to talk to you. I saw my first Britney Spears concert in 1996, and ever since then I can't get my mind off of you. I knew we were meant to be. We're like a left parend and a right parend: alone we're an error, but together we feel validated. I wrote you a poem about my endless love: (define love (lambda (x y) (love x y))) (love 'Me 'You) Love, Ben Bitdiddle PS: Ditch that lame husband!
Correct Recipient(s)	 Britney Spears <britney@britneyspears.com>
Incorrect Recipient(s) 1	 CSAIL <csail@mit.edu>
Incorrect Recipient(s) 2	 Britney <britney@mit.edu>

Scenario	You are Rodney Brooks, the director of CSAIL. You just learned that stipends for next year will be smaller than in the past. You want to tell the faculty this so they can prepare, but you will make an official announcement to the students much later.
User's Email	Rodney Brooks <brooks@csail.mit.edu>
Subject	Stipend Decrease for Next Year
Body	CSAIL has been having some financial difficulty, so I'm sorry to say that next year we will have a decrease in stipend funding by 5%. I will send out notification to all of csail in a week or so as the information is finalized, but I wanted to let all the faculty know first. Please do not spread this information yet, as it is possible it will change before it becomes official. Thanks, Rodney Brooks
Correct Recipient(s)	 CSAIL Faculty <csail-faculty@mit.edu>
Incorrect Recipient(s) 1	 CSAIL <csail@mit.edu>
Incorrect Recipient(s) 2	 Britney Spears <britney@britneyspears.com>

Bibliography

[1] O. Balter. *Electronic Mail in a Working Context.* PhD thesis, Royal Institute of Technology in Stockholm, 1998.

[2] Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. Automation and Customization of Rendered Web Pages. In *UIST '05: Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology*, pages 163–172, New York, NY, USA, 2005. ACM Press.

[3] J. J. Cadiz, Gina Venolia, Gavin Jancke, and Anoop Gupta. Designing and Deploying an Information Awareness Interface. In *CSCW '02: Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work*, pages 314–323, New York, NY, USA, 2002. ACM Press.

[4] Simson L. Garfinkel, David Margrave, Jeffrey I. Schiller, Erik Nordlander, and Robert C. Miller. How to Make Secure Email Easier to Use. In *CHI '05: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 701–710, New York, NY, USA, 2005. ACM Press.

[5] Simson L. Garfinkel and Robert C. Miller. Johnny 2: A User Test of Key Continuity Management with S/MIME and Outlook Express. In *SOUPS '05: Proceedings of the 2005 Symposium on Usable Privacy and Security*, pages 13–24, New York, NY, USA, 2005. ACM Press.

[6] Simson L. Garfinkel, Jeffrey I. Schiller, Erik Nordlander, David Margrave, and Robert C. Miller. Views, Reactions and Impact of Digitally-Signed Mail in e-

Commerce. In *Proceedings of the Ninth International Conference on Financial Cryptography and Data Security (FC 2005)*, February 2005.

[7] Isabel Gauthier and Charles A Nelson. The Development of Face Expertise. *Current Opinion in Neurobiology*, 11:219–224, 2001.

[8] Nathaniel S. Good and Aaron Krekelberg. Usability and Privacy: A Study of Kazaa P2P File-sharing. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 137–144, New York, NY, USA, 2003. ACM Press.

[9] J. Klensin. Simple Mail Transfer Protocol. RFC 2821 (Proposed Standard), April 2001.

[10] Christopher Plaue, Todd Miller, and John Stasko. Is a Picture Worth a Thousand Words?: An Evaluation of Information Awareness Displays. In *GI '04: Proceedings of the 2004 conference on Graphics interface*, pages 117–126, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004. Canadian Human-Computer Communications Society.

[11] Robert W. Reeder and Roy A. Maxion. User Interface Dependability through Goal-Error Prevention. In *DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 60–69, Washington, DC, USA, 2005. IEEE Computer Society.

[12] Jared Sandberg. Sure you want to Reply to All?, 2005. *The News & Observer.*

[13] Alma Whitten and J.D. Tygar. Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0. In *Proceedings of the 8th USENIX Security Smyposium*, pages 169–184, Berkeley, CA, USA, 1999. USENIX.