

FLANNEL: Adding computation to electronic mail during transmission

Victoria Bellotti[†], Nicolas Ducheneaut^{†‡}, Mark Howard[†], Christine Neuwirth*,

Ian Smith[†], Trevor Smith[†]

[†]Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304, USA
+1 650 812 4000
{bellotti, nicolas,
markhoward, iansmith,
tfsmith}@parc.com

[‡]School of Information
Management and Systems
Berkeley 102 South Hall
University of California
Berkeley, CA 94720-4600
nicolas@sims.berkeley.edu

*Departments of English and
of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
+1 (412) 268-8702
cmn@cs.cmu.edu

ABSTRACT

Keywords

INTRODUCTION

Email has become a central element of the way work is conducted in organizations where computers are used. More than just a simple communication medium [8], it is now the source of many different office tasks, serving as the place in which work is received and delegated [9]. For today's computer user at work, email is therefore much more than an ordinary application: it has become a *habitat*, the place where many people spend much, if not most, of their workdays [2,5]. Yet, despite the changing role of email for computer-supported work, the technical capabilities of this medium have remained surprisingly stable since its inception. Indeed, more than thirty years after the advent of email systems, most users still view a simple list of (generally plain ASCII) messages in chronological order and select from among those messages which ones to view in more detail. This seems strange when considered in the context of Moore's Law and the fact that the computing systems used to view and interact with email are easily two orders of magnitude more powerful than those originally used for email. Contrast this with the evolution of word processing applications over the last thirty years.

We felt that developing new email systems that applied computation to support the management of rapidly multiplying email tasks of users would ease the burden placed on today's email users. Two strategies for applying computation to email became immediately apparent:

- Improve the endpoints of communication, the email program in the user's hands.
- Improve the communication medium itself, by changing the protocols and underlying infrastructure used to manipulate email messages as they are transmitted.

This paper focuses on the second approach: *making the channel of email transmission "smarter."* By analogy, this approach has been successful for telephone companies who have long been in the business of adding more and more features to the communication channel they control (e.g. 'call waiting' and 'caller id' services) while still using endpoints whose technology has changed little in decades. In the case of email, this implies the modification of email servers, routing instructions, and protocols in an effort to provide new or better email services.

There is a substantial benefit to this second approach. On the surface, it preserves email's current agnosticism regarding the particular tools that will be available at the endpoints, i.e. it's not important to know what email client the recipient is using when you send a message. At a deeper level, the channel approach overcomes the barrier of user adoption by not requiring heavily invested email users to adopt new email clients.

We start below with a few examples of the potential applications one could develop if a smarter email channel was available. We then outline the architecture of our system, FLANNEL, which makes the development of such applications possible. After describing our current implementation and some of its design constraints, we finally conclude with a discussion of the pros and cons of our approach and how it could influence future research.

EXAMPLE APPLICATIONS

To demonstrate some services that could be provided by an enhanced email system using the channel-based approach, we will briefly describe some example applications below.

- Translation. User Alice sends a message to user Bob written in English. When the message arrives in Bob's inbox, it contains a French translation of the original text.
- To-do list. User Alice can send a message to user Bob, asking him to perform some action. The message is added to a web-based to-do list that keeps a record of outstanding items Alice is expecting. Alice periodically gets a status report of progress on these actions.

- Dynamic signature. On any message that Alice chooses to send, a randomly selected quote of the day is appended.
- Keyword searches. When Bob receives mail from Alice, appended to the message are the results of some web searches based on keywords found in the original message.
- Sales force automation. When a sales manager sends a qualified lead to a field sales office, a process is started around that lead. This process allows the sales manager to track the actual sales (conversion rate) of particular

buy-in from collaborators, no matter what their local set-up. In the following sections we show how this can be accomplished.

ARCHITECTURE

Figure 1 shows the major architectural components involved in performing additional computation on an electronic message during its transmission from sender 'Alice' to recipient 'Bob'.

For illustrative purposes it is assumed that the computation that Alice requires is language translation, i.e. Alice wishes

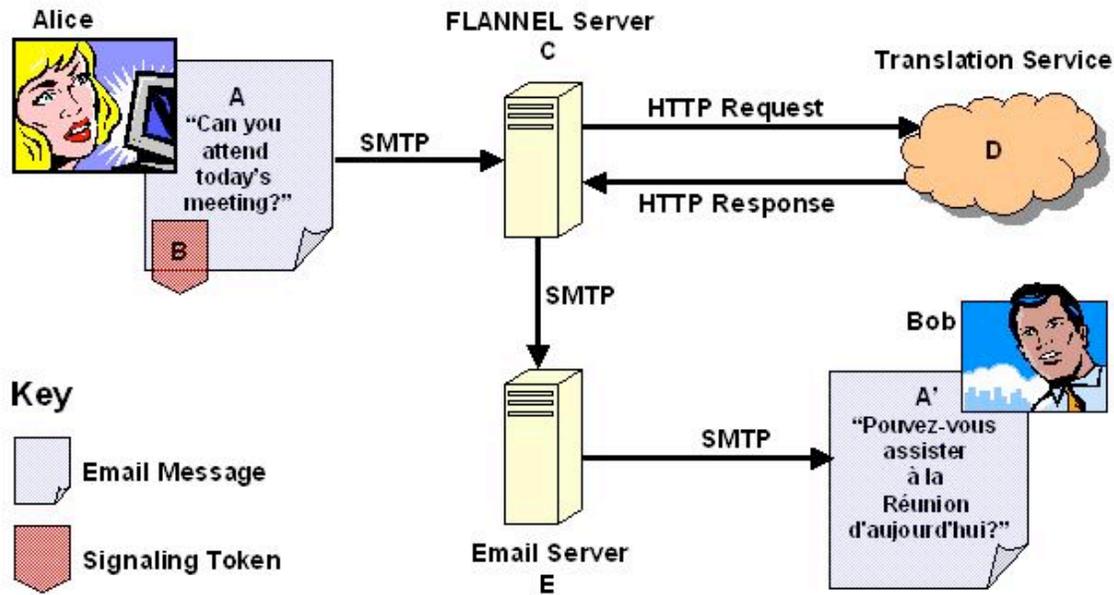


Figure 1: Overview FLANNEL Architecture

sales people when given qualified leads.

These kinds of services could be provided by local infrastructures such as, for example, task-management or leads-tracking software. But many solutions of this nature are peculiar to the user's own organization and do not lend themselves to cross-organizational collaboration. Shared virtual workspace applications (many of these can be found on the internet) might offer a possible alternative for some of these applications, but these require buy-in from all the collaborators and some set-up investment. However, by adopting a pure email channel-based approach we are able to provide solutions that work for anyone without requiring

to send an email message written in English to her colleague Bob and have that message translated into French during transmission. Figure 1 provides an overview of the architecture of 'FLANNEL,' our system for supporting services in the email channel. Messages (A) with a signaling token (B) are diverted to our server (C), which then passes appropriately parsed message content to a translation service (D) on the public internet. In reality, any computation service could be provided by (D) as long as the service provider speaks the appropriate protocols described below.

It should be noted that the FLANNEL system does not itself perform the translation of Alice's email message;

rather FLANNEL is a bridge between Alice's email and a third-party translation service, such as the AltaVista™ Babel fish service (<http://world.altavista.com>).

The steps involved in using FLANNEL to add computation to a message are separated into three phases, 'Signaling', 'Service Request', and 'Response Processing'.

Signaling Phase

In order to accomplish her goal of translating message (A) into message (A'), Alice must add some signaling to her message by which the FLANNEL system can discern two important pieces of information; namely who is going to perform the computation she desires (i.e. the service provider), and what data will the service providers require from FLANNEL in order to perform this service (i.e. the computation parameters). In the FLANNEL architecture these twin goals are accomplished by adding a single signaling token (B) to the initial message (A). The signaling token used in the current implementation of FLANNEL comes in the form of an 'internet shortcut' file, a text file containing a single URL. The URL contained in this file can thus meet the dual requirements of a signaling token by encoding the required service provider in the URL's host, and the required computation parameters in the URL's query string. For example, a simplified version of the URL which Alice adds to her message might be '<http://world.altavista.com?language=French>' where 'world.altavista.com' is the service provider, and 'language=French' is the single computation parameter. The precise format of a URL-based signaling token is discussed in greater detail below.

Assuming that Alice has previously obtained an internet shortcut file containing an appropriately formatted URL, then the operation of adding this signaling token to message (A) can be as simple as dragging and dropping the file onto her email message.

Once Alice has added the required signaling to her email message she simply sends the message in the normal manner, via her FLANNEL-enabled email server (C). We discuss how this diversion to the FLANNEL server occurs in the 'Implementation' section.

Service Request Phase

Upon receipt of an email message, a FLANNEL server (C) must first determine whether additional computation has been requested for that particular message. In our example the existence of a URL-based signaling token (B) tells the FLANNEL server that further computation is indeed required before message (A) is to be sent on its way to Bob.

The FLANNEL server then parses message (A) into its MIME components and examines the URL signaling token in order to extract the associated service provider and computation parameters. As described above the service provider can be extracted from a URL-based signaling token by examining the URL's host and the computation parameters can be found in the URL's query string. The FLANNEL server uses the computation parameters to produce an HTTP Request that is to be transmitted to the service provider. This HTTP Request can be thought of as

the 'function-call' that triggers the execution of the computation requested by Alice.

The computation parameters can come in two flavors, 'computation-specific' parameters determined by the Service Provider (D), or 'FLANNEL-specific' parameters governing the system's disposition toward the email message itself. Computation-specific parameters (e.g. 'Language=French') are not modified in any way by the FLANNEL server (C), they are simply added to the HTTP Request which the FLANNEL server sends to the Service Provider (D). FLANNEL-specific parameters, however, are processed by the FLANNEL server on receipt of the message to determine what message-specific information must be included in the HTTP Request. For example, Service Provider (D) must be sent the actual text of message (A) in order to translate that text into French, therefore the signaling token will include a FLANNEL-specific parameter of the form "TD-REQ-BODY=true". Detection of this computation parameter instructs the FLANNEL server to include the body text of message (A) in the HTTP Request being constructed for Service Provider (D). Should the Service Provider want to translate the subject line of message (A) in addition to the body text, then the signaling token would include the computation parameter "TD-REQ-SUBJECT=true" which instructs the FLANNEL server to include the message subject line in the HTTP Request being constructed. The precise details of the HTTP Request Protocol (and HTTP Response Protocol) can be found in the appendix to this document, which illustrates the available FLANNEL-specific parameters, and their effects.

Once the FLANNEL server has processed all the computation parameters found in the signaling token, and an appropriate HTTP Request has been constructed, this Request is transmitted to the Service Provider.

Response Processing Phase

When the Service Provider (D) receives the HTTP Request from the FLANNEL Server (C) it will carry out the translation of Alice's message and put the results of that translation into an HTTP Response, which is returned to the FLANNEL Server. The HTTP Response will contain a number of name/value pairs containing information about how to modify message (A). In our example, the Service Provider might send an HTTP Response containing a translation of the body and subject line for the recipient. In addition this response contains instructions for the FLANNEL Server about what to do with that data, for instance whether to replace the entire body text of message (A) with the French translation, or rather to merely append the French text onto the original text. The details of the HTTP Response Protocol are contained in the appendix of this document.

The HTTP Response will also contain an instruction for the FLANNEL Server as to what should be done with the modified email message (A'), i.e. whether or not to send the message onward to Bob. In our example the FLANNEL server will send the translated message to Bob via his email server (E), however there may be other

applications which would instruct the FLANNEL server to ‘swallow’ message (A) either indefinitely, or until some future date when it should be transmitted.

Once the FLANNEL Server has sent modified message (A’) on to Bob’s email server (E) the request for computation has been fulfilled, and Bob receives the translated email message in the normal manner. Note that Bob does not need to know about or make any investment in FLANNEL, to benefit from this service.

IMPLEMENTATION

Any implementation of the architectural model described above must address three key issues; routing, security, and interaction mechanisms. These issues are described in more detail below along with several strategies that can be used in response to the challenges they raise. Many of these strategies were developed and explored by the authors, and we will note which strategies we eventually settled on with our implementation of the FLANNEL architecture.

Routing

In order to perform a computation on an email message (A) a FLANNEL server must first receive that message. This means that while Alice may wish to send her message to Bob, if it is her intention to access a translation service, she must route her message first to the FLANNEL server (C) and only then on to Bob. This extended mail routing information can be conveyed in one of three ways.

1. Manual message modification by sender
2. Automatic server-side message interception
3. Automatic client-side message modification

In the first of these methods users address their messages to the FLANNEL server directly and include in the message (or it’s header fields) information about the ultimately intended recipient of the message. Thus the FLANNEL server is guaranteed to receive the message first and can take the appropriate action required to send the post-computation message on to Bob or any other recipient. We experimented with this technique by requiring that users modify the recipient line of a message in a simple and predictable way. We rewrote the addresses of our ultimate recipients (e.g. bob@some_domain.com) by replacing the “@” symbol with a special marker (e.g. “#”) and then appending the domain address of our FLANNEL server, so bob@some_domain.com would become bob#some_domain.com@flannel_server_domain.com. This simple modification technique allows users to accomplish the dual goal of routing a message to the FLANNEL server, and telling FLANNEL where to send it next in a single step. However, this technique has the unfortunate property of placing a significant burden on user’s to remember that they must change the ‘To:’ line of their messages, which could involve changing a large number of entries in their address book. For this reason we chose to discontinue our use of this approach.

The second routing method involves changing the behavior of a user’s email server such that it routes *all* of a user’s mail to FLANNEL first, along with the originally intended

recipient’s address. This approach removes the burden placed on a user described in the first approach, but replaces it with a potentially more serious one. That is, it can be extremely difficult to persuade the administrators of such a critical resource as an organization’s email server that it is a good idea to meddle with it in such a way as to intercept everyone’s email messages! We found this barrier to be insurmountable in our organization.

The third approach, that of automated client-side message modification, is the one we finally settled on in our implementation. Here the client software used to send and receive messages is modified to carry out the re-direction of mail to a FLANNEL server. We wrote a script which could modify the recipient addresses on a message in the manner described above for outgoing messages sent via the ubiquitous Microsoft® Outlook™ email client. The difficulty with this approach is that not all email users are Outlook™ users so in order to support a large range of users it must be possible to carry out these kinds of modifications in a large number of email clients.

It would be preferable to not alter the sender’s email client—so as to preserve the client-agnosticism of the email channel. Ideally, it would be possible to use the second approach above and intercept all email traffic. This would be possible in a large-scale commercial deployment of FLANNEL. However, the pragmatic problems of doing experiments in a production email environment made the third approach a reasonable trade-off.

It should be noted that, in each of these, once the initial message ‘capture’ has taken place, the FLANNEL server can re-write the ‘reply-to’ field of a message so as to automatically capture future replies by replacing the original sender’s email address with the email address of the FLANNEL server. This capture of future replies is independent of the server or client software used to generate those reply messages.

Security

Since the Service Provider (D) which FLANNEL communicates with in order to perform computation on a message may lie in a domain outside both that of the sender and of the FLANNEL server it is necessary to consider some security issues when implementing a FLANNEL system. Note, we are not referring to security here in the sense of encrypting data sent between a FLANNEL server and a service provider¹, rather we are referring to placing limits on the access service providers can have to the email messages of FLANNEL users.

The user of a FLANNEL system must be able to restrict the capabilities of a service provider in two ways, namely what parts of a message a service provider can read, and what parts of a message a service provider can write over (or append to). By default our system restricts service providers to having read access to the header fields of a message (e.g. sender, time stamp, recipient, and subject)

¹ Although this would certainly be an important capability of a commercial FLANNEL system

but not the message body, and gives append-only access to the message recipients and body. By overriding these default settings a user can allow particular service providers the ability to read the body of messages, add or replace attachment files, replace body text or recipient addresses and so forth.

These security privileges refer to the level of interaction between user and service provider, but there can also be issues regarding the level of interaction permitted between the FLANNEL system and a service provider. For instance it can be assumed that a FLANNEL server could potentially gather a significant amount of aggregate data regarding the email behavior of its users, controlling the amount of this data that a service provider can access will also be an important question for a commercial FLANNEL system. In our implementation we did not pay close attention to this issue and gave all service providers access to our repository through a network interface².

Interaction Mechanism

The question at issue here is what kind of interaction must a user employ in order to actually *use* our system? This question relates primarily to the way in which a user adds a signaling token to a message, since the question of how to route a message to FLANNEL has been addressed above. Broadly speaking there are three possible approaches to this problem.

1. Place tokens in the subject, recipient or other header field of the message.
2. Place tokens, code or other markup in the body of the message.
3. Add one or more attachments to the message.

In seeking to choose between these strategies we sought to try and preserve the simplicity and ease of use inherent in existing email practices. We felt it was extremely important that the act of adding signaling information to a message should be both simple and familiar to email users.

It is worth restating at this point that the signaling phase of a FLANNEL interaction must accomplish two goals. The first is to establish the identity of a service provider, and the second is to convey any computational parameters required by a FLANNEL server or a service provider in order to perform a computation³. In early implementations of the FLANNEL system we adopted a signaling strategy based on the first approach above. It was necessary to send to pre-configured email addresses that corresponded to particular computation services. This technique falls foul of the same problems seen in the discussion of routing above, namely that it forces users to remember a large number of potentially complex email addresses, e.g. add-

² It should be noted that all service providers used by our system were actually servers built by the authors and resided on our own network.

³ One of these computational parameters will by necessity be an indication of which particular computation is to be performed.

task@flannel.com. Further, this strategy becomes increasingly more complex as computations requiring more parameters are made available since there must be pre-configured email addresses for all combinations of parameters, e.g. reminder-in-2days@flannel.com, reminder-in-5days@flannel.com, etc.⁴

We felt that many of the problems encountered when adopting this signaling strategy would occur again should we adopt a strategy of placing code or markup in the body of a message. It is certainly the case that this would place a significant burden on a user of such a system as they would have to learn and use a complex language while writing their email messages. For this reason we did not choose to implement a markup style signaling strategy.

The final strategy available was to attach files to a message containing the required signaling information. This type of approach has the benefit of allowing users to use an extremely familiar interaction mechanism in order to accomplish the signaling phase, i.e. they could use whatever attachment interaction they were already familiar with, whether that be dragging files from their desktop to a message, or using menu's or hotkeys to attach the same files. The authors considered two kinds of attachment based signaling strategy during the implementation phase of this project. Initially it was believed that this approach could best be carried out simply by adding text files to messages as attachments, the contents of which would be name/value pairs corresponding to the computational parameters required by the system. This strategy was not actually implemented since we quickly came upon what we believed to be a far superior strategy. This second signaling mechanism was based on the principle of adding "internet shortcut" files as attachments. These files are actually just text files containing a single URL. These URLs can easily meet the dual goals of a signaling token by encoding the service provider in the URLs host, and the computation parameters in the URLs querystring.

Using internet shortcut files as signaling tokens gave us several distinct advantages over simple text files⁵. First, it is possible to create these shortcut files by simply dragging an icon or a link from Microsoft's Internet Explorer™ onto the desktop. Secondly, clicking on one of these files opens the URL in Internet Explorer. A sophisticated service provider can customize the web page presented to allow FLANNEL-specific interactions.

The URL based signaling strategy also allowed us to turn all existing web sites into simple FLANNEL service

⁴ We attempted to move some signaling information from the recipient line to other header fields, such as x-headers. However, this created other problems since as messages crossed domain boundaries it appeared that the header information could be lost or changed in unpredictable ways.

⁵ Many of these benefits rely on behaviors peculiar to the MS Windows operating system and MS Internet Explorer, but it is our belief that they could be reproduced in other environments.

providers instantly since it was relatively simple to define default behavior such that, in the absence of other FLANNEL specific parameters, any URL could be simply requested by the FLANNEL server and appended to a message. It also allows developers of FLANNEL based applications to use existing web application development tools in order to generate FLANNEL applications. For instance all applications developed by the authors were implemented using Java servlets. These servlets would respond to an HTTP GET request by displaying a web page where the parameters of the computation could be manipulated (this corresponds to a user double clicking on an internet shortcut file) while HTTP POST requests would actually engage the computation defined in those parameters.

We tried variations of each of these approaches in a series of tests in which test users from among our own team, and later other users, tried the system over a period of months. Indeed, one overloaded manager used the system over a period of several weeks. Our test application was a simple project management tool that allowed users to assign actions, deadlines and reminders for themselves and others and that provided a daily status report in email that contained links to the FLANNEL server's web interface (where certain users could carry out certain operations such as viewing status and customizing their daily report).

APPLICATION DESIGN CONSTRAINTS

In the process of building the applications described above the authors gained significant insights into what properties a successful FLANNEL-based application would have. It is our belief that any application that has the three properties outlined below will be both useful and compelling when implemented so as to take advantage of the FLANNEL architecture.

1. Send time and computation time are asynchronous.

The sender cannot be interactively prompted for more information, nor can the computation be interrupted. The computation parameters are exclusively the URL, the email, and the state of the network at the time computation is to be performed. For example, if there is difficulty translating the message body into French, the application cannot query the sender for disambiguation without returning the original mail.

This is distinctly different from applications built into email clients (such as Outlook's task management functionality) that trigger computation such as input validation and synchronization when sending, reading, and retrieving email.

2. Sender chooses the computation.

In FLANNEL, the user cannot trigger computation without sending email. In situations in which the user wishes to trigger computation with no other party involved, she must email herself or use an alternative interface such as a web browser.

Additionally, this constraint has the effect of requiring the sender to know all information for the computation in

advance. In the case of the "Translate to French" example, the application is unable to query the recipient on their language preference before delivery of the translation; Alice must determine that Bob would prefer to receive her message in French.

3. Sender does not require the output of the computation.

For applications with unpredictable outputs, such as the occasional faux pas in the "Translate to French" application's translation engine, the sender will not be queried for confirmation before the translation is received.

POSSIBLE APPLICATIONS

We now return to the scenarios we outlined at the beginning of this paper to give some sense of how the FLANNEL system manages them and to discuss its potential for integration with existing infrastructures. All of the applications mentioned below have been constructed in whole or in part.

Tracking Action Status: In our To-do list scenario, Alice sends a message to Bob, asking him to perform some action. The message is added to a web-based to-do list that keeps a record of outstanding items Alice is expecting. Alice periodically gets a status report of progress on these actions.

Solution: FLANNEL extracts the action(s) from the message, which are either flagged in the message body, or included in a template form that FLANNEL can parse. Bob receives a message with the action items translated into links that he can click on to update the to-do list maintained on a 3rd party server. While our test implementation was able to provide just such a service in a stand-alone manner, it is easy to see how this mechanism could be integrated with any more powerful intranet service that maintains to-do lists or tracks workflow to provide a more flexible way for users to update status without having to use a standard web interface.

Dynamic signature: On any message that Alice chooses to send, a randomly selected quote of the day is appended.

Solution: FLANNEL takes any message with the appropriate signal and diverts it to a service that responds to requests for a "quote of the day," or similar text or graphical output (such as cartoons) from a database. This solution would also work in a corporate infrastructure for providing brief news updates, links, stock quotes and so on.

Keyword searches: When Bob receives mail from Alice, appended to the message are the results of some web searches based on keywords found in the original message.

Solution: FLANNEL parses the message body looking for uncommon words (perhaps users might sign up for a keyword list and specify a list of trusted sources for searches), which are submitted to one or more internet-based search engines. The results of the searches may be delivered as text or as links (possibly to a cache of the results stored temporarily on the FLANNEL server).

Sales force automation: When a sales manager sends a qualified lead to a field sales office, a process is started around that lead. This process allows the sales manager to track the actual sales (conversion rate) of particular sales people when given qualified leads.

Solution: FLANNEL parses the message and diverts text from the body (or possibly attached files) to a leads-tracking service. The body of the message (or attachment) containing the lead(s), is replaced with one or more URLs pointing to the leads-tracking service. If the field salesperson does not click on a link to obtain the lead, the sales manager will know (by email notification or by accessing a web site) that it has not been used and must be reassigned after a specified period. If the salesperson clicks on the lead, a timer can track how long it takes to convert that lead into a sale (this might allow the manager to gauge employee performance or understand which leads are more difficult to convert). This solution could be integrated with a web-based leads-tracking service to offer an alternative means of interacting with its database (normally users would access web pages directly to exploit this kind of functionality).

In each of these examples it should be noted that the FLANNEL service offers an alternative mechanism to the tedious process of opening a web-browser and navigating to a service interface to execute functionality. Existing internet and intranet services are made more easily available from *inside the email habitat* where users tend to spend a great deal of their work-life.

RELATED WORK

Most of the earlier work on computational mail, unlike FLANNEL, relied on the *endpoints* of a communication system to perform the computation. The earliest mention of such research can be found in the description of the RAND Intelligent Terminal Agent (RITA) [1], in which programs are embedded in each message. When the message is opened, the program is executed (in a fashion strikingly similar to today's email viruses, and therefore causing potential security problems). One frequently cited application of these embedded programs is collaborative decision making, such as asking a number of recipients questions about suitable times for a meeting and centrally collecting them. Borenstein [4] later described Atomicmail, a more secure version of computational mail allowing its users to build CSCW applications on top of email with a LISP-based language. Borenstein's Atomicmail in fact extends some ideas already present in an earlier system, the Andrew Message System (AMS) from Carnegie Mellon University [3], that he contributed to. In AMS users could already send voting messages, return receipts requests, enclosures, or subscription invitations. The recipient then interacted with these messages via dialog boxes, menus and check boxes. AMS also supported messages including text, pictures, or animations, making it one of the earliest multimedia mail systems, long before HTML mail. But as we mentioned earlier however, the email channel itself remained unchanged – the endpoint; the user's email software, was the only place where computation took place.

Active Mail [7] is another system with “enhanced endpoints.” It addresses two issues: first, that users find it hard to maintain dialogue continuity in email, and second that they find it hard to maintain document consistency. In Active Mail, interactive messages support interactions between sender, receiver and future participants in a conversation. The messages are treated as a shared space that the participants can edit from within their own email client, in a fashion reminiscent of the more recent, commercial system called Zaplets [10].

Compared to the systems we have just described, the work on Envoys [6] employs a channel-based strategy much more similar to FLANNEL, but which does not use the well-established and well-understood SMTP and HTTP protocols to accomplish the goal of accessing computational resources through email messages.

Envoys are electronic mail messages that make requests to remote electronic mailers. They differ from conventional messages in that they might be routed to recipients that have not been specified, their requests can often be carried out with no human intervention, they can be modified as they move from mailer to mailer, and they return to the original sender to inform her what actions have been taken. Envoys also provides a reasoning system and a language for communicating with the mailer (MCL, Mail Control Language) that users can insert in the form of scripts in the body of their messages. The Envoys system is implemented using mailers based on a specific protocol, ETC (Elapsed Time Communication) – which is probably why it was never widely deployed, since changing the internet's core infrastructure is notoriously hard.

CONCLUSIONS

In this paper, we have presented a mechanism to allow senders of email to access computational resources that are embedded in the email channel itself. We facilitate this access to computational resources through the protocols and infrastructure of the Web. Therefore, the tools and knowledge of the existing community of Web developers can be leveraged to develop email-based applications. In conjunction with this web-based approach, we have developed a UI technique that is simple for users and powerful for developers—using internet shortcut files as signaling tokens. We have proposed a set of guidelines to aid the evaluation of potential applications to be deployed in the email channel.

FUTURE WORK

At present, FLANNEL treats the body of an email message in a relatively simple-minded way. Little if any attempt is made to derive structure from the message body. Applications that require structure in the body (e.g. ASCII forms or natural language understanding) must provide that functionality. For example, the “Translate to French” application cannot easily determine that there are portions of the message that should not be translated. We propose that a future implementation of the FLANNEL architecture might include some standardized message body parsing machinery to address this weakness.

REFERENCES

1. Anderson, R., & Gillogly, J. (1976). Rand intelligent terminal agent (RITA): design philosophy (R-1809-ARPA, February 1976): RAND Corporation.
2. Bellotti, V., Smith, I. (2000). Informing the Design of an Information Management System with Iterative Fieldwork, *Proceedings DIS'00 Symposium on Designing Interactive Systems*.
3. Borenstein, N., & Thyberg, C. (1988). Cooperative work in the Andrew message system, *Proceedings of the conference on computer-supported cooperative work* (pp. 306-323). September 26 - 28, 1988, Portland, OR USA.
4. Borenstein, N. (1992). Computational mail as network infrastructure for computer-supported cooperative work, *Proceedings of the CSCW92 conference on computer-supported cooperative work*. November 1 - 4, 1992, Toronto Canada.
5. Ducheneaut, N., & Bellotti, V. (2001). Email as habitat: an exploration of embedded personal information management. *Interactions*, 8(5), 30-38.
6. Gold, E. (1986). Envoys in electronic mail systems, *Proceedings of the third ACM-SIGOIS conference on office automation systems* (pp. 2-10). October 6 - 8, 1986, Providence, RI USA.
7. Goldberg, Y., Safran, M., & Shapiro, E. (1992). Active mail - a framework for implementing groupware, *Conference proceedings on computer-supported cooperative work* (pp. 75-83). November 1 - 4, 1992, Toronto Canada.
8. Mackay, W. E. (1988). More than just a communication system: diversity in the use of electronic mail, *Proceedings of the conference on computer-supported cooperative work*. September 26 - 28, 1988, Portland, OR USA.
9. Whittaker, S., & Sidner, C. (1996). Email overload: exploring personal information management of email, *Conference proceedings on Human factors in computing systems* (pp. 276-283). April 13 - 18, 1996, Vancouver Canada.
10. Zaplets. (2001). Zaplets. Available: <http://www.zaplet.com/>.

APPENDIX I: FLANNEL REQUEST PROTOCOL

The following are the names and values of parameters sent to service providers in the FLANNEL architecture, see previous "Signaling Phase" section for more details.

TD-REQ-SENDER

If this parameter is set to TRUE in a command URL then this parameter will appear in the resulting Request object containing the email address of the sender of the original email message.

TD-REQ-RECIPIENTS

If this parameter is set to TRUE in a command URL then this parameter will appear (one or more times) in the resulting Request object containing the email address(es) of the recipient(s) of the original email message.

TD-REQ-SUBJECT

If this parameter is set to TRUE in a command URL then this parameter will appear in the resulting Request object containing the subject line of the original email message.

TD-REQ-DATE

If this parameter is set to TRUE in a command URL then this parameter will appear in the resulting Request object containing the date of the original email message.

TD-REQ-MESSAGEID

If this parameter is set to TRUE in a command URL then this parameter will appear in the resulting Request object containing the FLANNEL generated ID of the original email message.

TD-REQ-HEADERS

If this parameter is set to TRUE in a command URL then all SMTP headers found in the original message will appear in the resulting Request object.

TD-REQ-BODY

If this parameter is set to TRUE in a command URL then the body of the original email message will appear in the content stream of the resulting Request object.

TD-REQ-THREAD

If this parameter is set to TRUE in a command URL then this parameter will appear in the resulting Request object containing the FLANNEL generated Thread ID of the original email message.

TD-REQ-ATT-LIST

If this parameter is set to TRUE in a command URL then this parameter will appear in the resulting Request object containing the names of any attachments on the original email message.

APPENDIX II: FLANNEL RESPONSE PROTOCOL

The following are the names of response parameters that can be used by service providers to request some action to be taken on their behalf by the FLANNEL server. For more information on these parameters, please see the section on "Response Processing Phase" above.

TD-RES-REPLACE-SENDER

If this parameter appears in a POST Response then the FLANNEL service will take the value of this field and use it to modify the sender of the original email message.

TD-RES-ADD-RECIPIENT

If this parameter appears in a POST Response then the FLANNEL service will take the value of this field and use it to modify the recipients of the original email message.

TD-RES-REMOVE-RECIPIENT

If this parameter appears in a POST Response then the FLANNEL service will take the value of this field and use it to modify the recipients of the original email message.

TD-RES-REPLACE-SUBJECT

If this parameter appears in a POST Response then the FLANNEL service will take the value of this field and use it to modify the subject of the original email message.

TD-RES-APPEND-BODY

If this parameter is set to TRUE in a POST Response then the FLANNEL service will take the content stream of this Response and use it to modify the body of the original email message.

TD-RES-REPLACE-BODY

If this parameter is set to TRUE in a POST Response then the FLANNEL service will take the content stream of this Response and use it to modify the body of the original email message.

TD-RES-ADD-ATT

If this parameter is set to TRUE in a POST Response then the FLANNEL service will take the content stream of this Response and use it to modify the attachments of the original email message.

TD-RES-REMOVE-ATT

If this parameter is set to TRUE in a POST Response then the FLANNEL service will take the content stream of this Response and use it to modify the attachments of the original email message.

TD-RES-REMOVE-CMD

If this parameter is set to TRUE in a POST Response then the FLANNEL service will remove the command URL attachment from the original email message.

TD-RES-SWALLOW-MSG

If this parameter is set to TRUE in a POST Response then the FLANNEL service will not proceed with sending this message to its originally intended recipients.

The columns on the last page should be of equal length.