Zest: Discussion Mapping for Mailing Lists

Ka-Ping Yee
Group for User Interface Research
University of California, Berkeley
ping@zesty.ca

ABSTRACT

Structured mapping and coordination techniques are effective for making sense of conversations and building organizational memory. This paper proposes a new method for processing e-mail that brings these benefits to mailing lists. The technique leverages existing e-mail conventions instead of requiring participants to communicate within an imposed framework. Zest, a prototype browser for e-mail discussions, processes a standard e-mail folder, and displays a concise, conversation-like overview of the discussion in progress. In effect, Zest transforms an ordinary mailing list into a structured conversation system, without the need for special client software, and without the rigid formalism that previous such systems forced upon their users. Informal evidence suggests that the overviews shown by Zest are more readable and effective than the threaded message views that e-mail clients currently provide, and also that Zest stands a better chance of adoption than previous structured conversation tools.

Keywords

Electronic mail, threading, critical discussion, argument visualization, organizational memory, decision support, issue-based information systems.

INTRODUCTION

E-mail is the single most prevalent computer-mediated communication medium. Mailing lists are a popular CSCW tool for long-term discussion, probably because they are simple to understand, they require no special client software or training, and they are straightforward to administer. Several mailing list management programs such as Mailman, Listserv, and Majordomo are available for free and are relatively easy to set up; free list hosting services such as Yahoo! Groups are also available on the Web.

Electronic mailing lists are in extremely widespread use. In July 2000, for example, eGroups claimed to have 17.5 million users [6]; today it is part of Yahoo! Groups, which claims to offer “hundreds of thousands of e-mail groups”. Typical lists have somewhere from a dozen to a few hundred subscribers, and particularly active lists can generate a hundred or more messages a day. Organizing and reading all of these messages can be a significant problem, and it is common to hear colleagues comment that they feel overwhelmed by mail they get from mailing lists.

Often mailing lists are used as an informal coordination and decision-making venue for teams in the workplace, geographically distributed software development groups, or technical standards working groups. In such contexts, the archived discussion can be an especially important resource for maintaining organizational memory. Participants can use the archive to catch up on missed messages, to determine the resolution of an open issue, to find the rationale for a particular decision, or to answer questions of their own that have already been asked by others. However, existing archiving tools for e-mail are not as strong as they might be in supporting these tasks, and more sophisticated visualization tools are not very widely used.

This work aims to improve the ability of mailing lists to serve as an organizational memory and decision support medium, while helping users deal with high mail volume.

RELATED WORK

E-mail and newsgroup client programs embed standard RFC822 headers into messages to link replies. When the user composes a reply to another message, the Message-ID of the original message is placed in an “In-Reply-To” header on the reply message. Newsgroup readers, mail archiving tools, and e-mail client programs use this information to produce a threaded view consisting of an outline list where each message is represented by its subject line, and the subject lines of replies are indented beneath the messages to which they reply. The use of these message-to-message reply links and the outline-style presentation of message threads is practically universal among all known popular mail processing software.

There have been many projects to date in visualizing textual discussions, some of which go beyond the e-mail domain. Threaded Chat [10] lets users manually attach individual chat messages as replies to others, so that a conversation becomes an evolving tree rather than a scrolling list of messages. Chat Circles, Conversation Landscape, and Loom [2] give an overview of behavioural patterns, but do not show content in a way that would support collaborative tasks; they are intended to be social visualizations rather than organizational memory tools. Netscan [9] provides several visualizations for newsgroup discussion structure, most notably time-based thread tree and piano roll views, but none of its visualizations display any message content. An advanced e-mail client under development at IBM [8] also includes a time-based thread tree without message content; for showing content, the client displays miniaturized views of entire messages. Thumbnails can be helpful for finding a previously seen message, but not for extracting the outcome of a discussion.

Past efforts to identify and visualize conversation structure include The Coordinator and IBIS. The Coordinator [4] drew on speech act theory to organize conversations in terms of requests, commitments, offers, and so on. Deployment failed in practice because users were unwilling to fit every message into the explicit categories that The Coordinator imposed. IBIS [7] organizes arguments into nodes, each of which is an issue, position, or argument, joined by eight types of links. IBIS has been implemented in a graphical tool, gIBIS [1]. Studies have observed that the overhead of specifying structure can be an obstacle to its use, and the tendency of discussions to be sidetracked by debates on the correct use of the IBIS structure [1] further suggests that the structure may be too rigid.

Existing tools, then, generally fall into two categories:

(a) those that work with existing e-mail or newsgroup discussions, hence requiring no change in practice, but yield relatively little conversational structure (only basic threading at a whole-message granularity);

(b) those that can build rich conversational structures, but require a large change in practice, as they do not work with e-mail at all and impose an entirely new discussion medium.

The goal of this work is to achieve the benefits provided by structured conversations without the pitfalls. The tools in category (b) actually suffer from two barriers to adoption: they ignore the installed base of e-mail software, and they impose extra work on users even after they have switched to new software. Zest avoids both problems, as it is designed to work with e-mail exactly as it is written today (and even with previously archived e-mail).

DESIGN

Methods for quoting text from other messages are fairly standardized, both in writing conventions and in e-mail client software. There are only a few major quoting styles, and almost all of them set off quoted text from the left margin by a column of “>” symbols, so quoted text is easy to detect. Zest divides each message into contiguous blocks of quoted or unquoted text called *sections*. Established e-mail writing conventions make it safe to assume that unquoted text immediately following a quoted section is a reply to the text of that quoted section.

Sections are then threaded according to these quote-reply chains. For messages that cannot be threaded according to quoted text, we fall back on whole-message threading based on message headers. Each connected tree of message sections then constitutes one thread. Although each section belongs to one thread, a message may belong to more than one thread if it replies to sections from different threads. The default main index of the message archive is a listing of threads by the date of most recent activity.

In the overview for an individual thread, each section is represented by the sentences in its first two lines. This gives a much more relevant overview of the content than simply repeating the subject line many times. To keep the summary concise, not all sections appear. Any section that replies to or is replied to by another section appears; also, the first section of any message not otherwise represented by some section appears.

The result is a remarkably natural point-counterpoint summary of the thread that reads like a conversation. Informal tests show that this technique can produce reasonable results from real e-mail written by authors with no knowledge of the threading algorithm.

OPTIONAL FEATURES

We allow users to easily turn their overview into an argument map using an optional typographic convention. By inserting a textual symbol called a *criticon* at the beginning of any paragraph, users can force a section break and tag the new section as one of four types: `[?]` marks a question, `[#]` marks a statement, `[+]` marks a supporting argument, and `[-]` marks an opposing argument. This four-type scheme provides most of the useful semantics of IBIS, but is simpler and easier to remember.

The special criticon `[!]` marks a section as an alleged resolution of a discussion and flags the entire thread *resolved* in the listing of threads.

In the thread summary, these types are shown with small coloured icons, but the criticons themselves remain visible in the text of messages, just as they were typed. The use of symbols typed into the message body, and their visibility in the messages when they are received and displayed, support *learning by example*, as highlighted below.

APPLICATIONS

Zest allows participants to quickly see which issues are resolved or open, and what questions have already been asked and answered. A glance at the colours of the icons displayed in the overviews can give a rough sense of the group’s opinion. Looking at the icons next to author names can suggest who best to ask for help on a given topic.

When a mailing list is used as a medium for long-term or distributed meetings, it can serve as a meeting-capture tool. In particular, Zest supports the capture of design rationale.

The structure provided by Zest can also improve group decision-making. A previous study has shown that groups are more likely to arrive at a consensus if their online interactions are structured [3]. Improved access to the content of previous discussions could help keep participants from rehashing old debates and repeating the same mistakes in reasoning, thus also increasing the group’s ability to reach consensus.

Because all of these benefits support the use of mailing lists for debate and critical discussion, public policy and e-democracy are exciting potential application domains; however, truly adversarial situations are not addressed here.

EXAMPLE

Figures 1, 2, and 3 illustrate the basic algorithm. Figure 1 shows two messages from the developers' mailing list for E, a secure programming language. Tyler's message suggests (a) a new design heuristic and (b) that there should be no separate "type" object. Chip's message responds to both suggestions. Figure 2 shows the outline produced from these two messages by standard threading (using Pipermail). Figure 3 shows the overview displayed by Zest.

After Chip's message, several more messages were written on this thread. In Figure 4, the Zest browsing interface shows the entire thread. For this figure only, criticons were added to the messages to demonstrate how they would be used in practice. Had the messages been left untouched, only the triangular icons in the left pane would be missing.

Notice that the overview (Figure 4, left pane) effectively captures the knowledge exposed by the discussion, including the answer to Tyler's suggestion and its rationale. In particular, it yields the answers to questions such as:

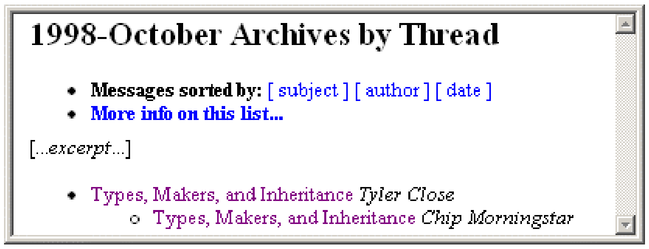
- Why are the type and maker objects separate?
- What design principles are being applied?
- Why are type objects necessary?

— yet no natural language processing was needed, and *no extra work* was required from the participants.

```
From: Tyler Close <tyler@lfw.org>
To: Ka-Ping Yee <ping@lfw.org>
Cc: e-lang@eros.cis.upenn.edu
Subject: Re: Types, Makers, and Inheritance
Date: Tue, 20 Oct 1998 02:37:46 -0400

[...excerpt...]

I would like to propose a new design heuristic. Anything that causes
Ping to make a diagram is too complex and must be simplified.

I would like the type object and the maker to be identical (ie: no
separate type object). I will also reiterate my desire to eliminate
inheritance.

Tyler
```

```
From: Chip Morningstar <chip@communities.com>
To: e-lang@eros.cis.upenn.edu
Subject: Re: Types, Makers, and Inheritance
Date: Tue, 20 Oct 1998 09:33:19 -0700 (PDT)

Tyler sez:
>I would like to propose a new design heuristic. Anything that causes
>Ping to make a diagram is too complex and must be simplified.

I think this is a fine heuristic, as long as our benchmark is Ping and
not MarkM. MarkM will make a diagram for anything :-)

>I would like the type object and maker object to be identical (ie: no
>separate type object). I will also reiterate my desire to eliminate
>inheritance.

I agree. While the separation of the type and maker has a certain
conceptual elegance, this does not IMHO come near to offsetting the
additional cognitive and notational overhead of having yet another
abstraction to keep track of, explain, code, etc.

Chip
```

Figure 1. Two actual e-mail messages from the E language developers' mailing list.



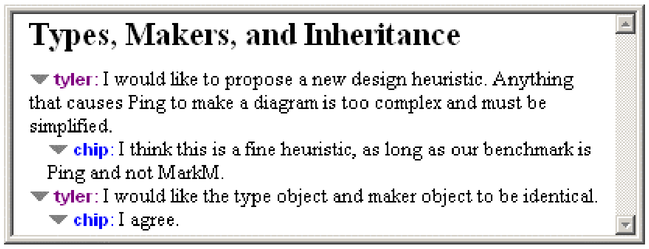
1998-October Archives by Thread

- **Messages sorted by:** [subject] [author] [date]
- **More info on this list...**

[...*excerpt*...]

- Types, Makers, and Inheritance *Tyler Close*
  - Types, Makers, and Inheritance *Chip Morningstar*

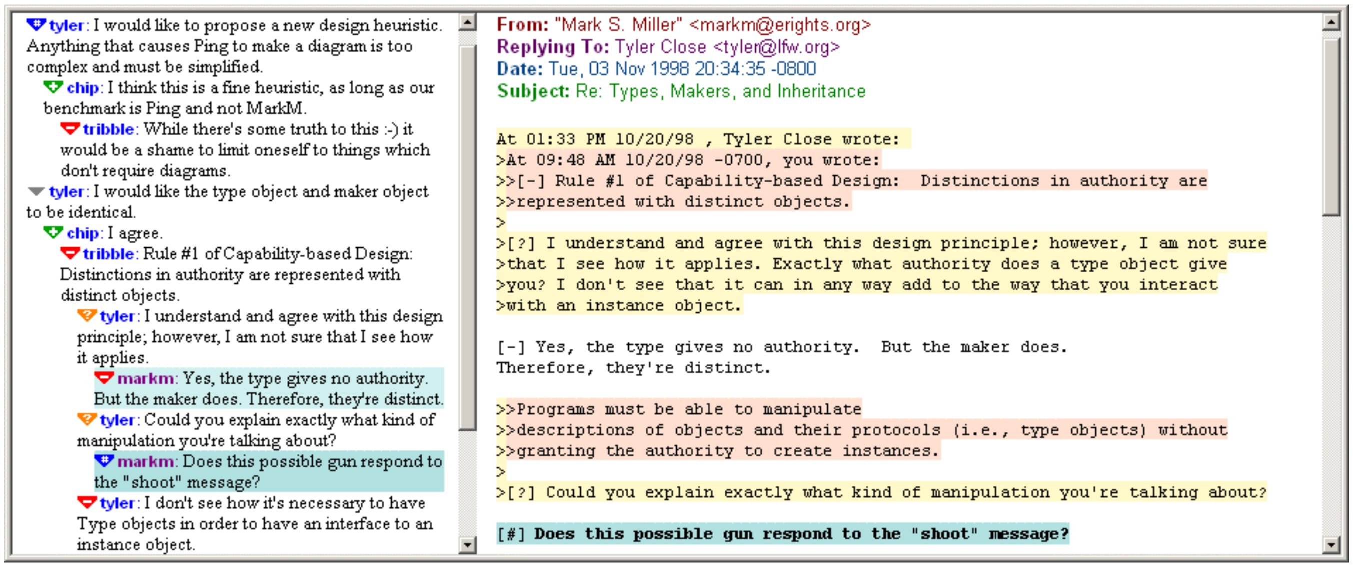
Figure 2. Standard subject-threaded view.



Types, Makers, and Inheritance

tyler: I would like to propose a new design heuristic. Anything that causes Ping to make a diagram is too complex and must be simplified.
  chip: I think this is a fine heuristic, as long as our benchmark is Ping and not MarkM.
tyler: I would like the type object and maker object to be identical.
  chip: I agree.

Figure 3. Zest content-threaded view.



tyler: I would like to propose a new design heuristic. Anything that causes Ping to make a diagram is too complex and must be simplified.
  chip: I think this is a fine heuristic, as long as our benchmark is Ping and not MarkM.
    tribble: While there's some truth to this :-) it would be a shame to limit oneself to things which don't require diagrams.
tyler: I would like the type object and maker object to be identical.
  chip: I agree.
    tribble: Rule #1 of Capability-based Design: Distinctions in authority are represented with distinct objects.
      tyler: I understand and agree with this design principle; however, I am not sure that I see how it applies.
        markm: Yes, the type gives no authority. But the maker does. Therefore, they're distinct.
      tyler: Could you explain exactly what kind of manipulation you're talking about?
        markm: Does this possible gun respond to the "shoot" message?
      tyler: I don't see how it's necessary to have Type objects in order to have an interface to an instance object.

From: "Mark S. Miller" <markm@erights.org>
Replying To: Tyler Close <tyler@lfw.org>
Date: Tue, 03 Nov 1998 20:34:35 -0800
Subject: Re: Types, Makers, and Inheritance

```
At 01:33 PM 10/20/98 , Tyler Close wrote:
>At 09:48 AM 10/20/98 -0700, you wrote:
>>[-] Rule #1 of Capability-based Design:  Distinctions in authority are
>>represented with distinct objects.
>
>[?] I understand and agree with this design principle; however, I am not sure
>that I see how it applies. Exactly what authority does a type object give
>you? I don't see that it can in any way add to the way that you interact
>with an instance object.

[-] Yes, the type gives no authority.  But the maker does.
Therefore, they're distinct.

>>Programs must be able to manipulate
>>descriptions of objects and their protocols (i.e., type objects) without
>>granting the authority to create instances.
>
>[?] Could you explain exactly what kind of manipulation you're talking about?

[#] Does this possible gun respond to the "shoot" message?
```

Figure 4. In this view of the entire thread, the user has selected Mark's reply (shaded, at bottom) to Tyler's question. The content-threaded outline generated by Zest, at left, provides a conversation-like overview of this exchange.

PHILOSOPHY

How is Zest able to produce more structure than previous mail processing tools, even when it processes exactly the same e-mail? It obtains extra information in two ways: it takes better advantage of existing e-mail conventions, and it offers the user ways to *optionally* add more structure.

Zest exemplifies a "gentle seduction" design philosophy[1] for CSCW applications: that is, the system behaves in familiar ways when used conventionally, but users can voluntarily take advantage of fancier features, and thereby reap direct rewards. Rather than confronting current practice with an extinction-level event, we encourage practice to gradually *evolve* from current practice towards a practice that benefits more fully from new features.

This philosophy emphasizes four properties:

- predictability
- backward compatibility
- ability to learn by example
- visible payoff

The threading algorithm is intentionally kept simple and predictable so that users can choose to write e-mail to get the effect they want (for example, placing the most important sentence first in a paragraph). It is backward compatible with current e-mail conventions so that users are not forced to learn new conventions right away. The new conventions are clearly visible and easy to understand so that users can learn by observing how other users apply the conventions. When newcomers try the extra features, their visible effects promote continued use.

The "gentle seduction" design philosophy is specifically aimed at addressing six of Grudin's eight challenges for groupware developers [5]:

- *predictability* simplifies exception handling and reduces the likelihood of intuition failure;
- *backward compatibility* helps to evade critical mass problems, and also alleviates the problem of designing for infrequently used features;
- ability to *learn by example* addresses the problem of acceptance;
- *visible payoff* reduces the perception of disparity between expender of effort and receiver of benefit, and helps relieve Prisoner's Dilemma problems.

EVALUATION

A natural next step is a user study. Although a formal user study of this tool has not yet been conducted, there is some evidence to believe that it has promise.

First, criticons have already successfully demonstrated the *learning by example* property. As an experiment, a few members of a mailing list, including myself, began inserting criticons into our normal postings as a way of calling attention to important points. Before long, several others had picked up the practice of using them; many did so without asking any questions, and all used them correctly. This took place even though there was no software tool analyzing the messages; people used criticons merely in order to clarify their messages to other people.

[1] "The Gentle Seduction" is the title of a short story by Marc Stiegler about the gradual adoption of radical new technologies.

Second, there is evidence from current practice to support the viability of manual typographic annotations. On some developers' lists, participants already write "`+1`" or "`-1`" to indicate their opinion on an issue. This suggests that people are willing to follow simple structural conventions.

Third, Figures 3 and 4 demonstrate that even if no one decided to annotate their messages, the resulting summary would still be more useful than the outlines produced by current tools. Most people were taught to begin paragraphs with topic sentences and already do so as a matter of habit.

Finally, Zest has been informally evaluated. When a summarized thread, generated from real, unedited e-mail taken from a real mailing list, was shown to an audience of about fifty developers, most responded positively and expressed interest in using Zest when it becomes available.

DEMONSTRATION

The presenter is the author of Zest. Attendees will be able to try out the Zest interface on a laptop to explore large online discussions. The demo requires only desk space.

ACKNOWLEDGEMENTS

The idea of threading messages by paragraphs and quotations instead of subject lines was first suggested by Terry Stanley in 1997. Mark S. Miller also contributed to this work.

REFERENCES

1. J. Conklin, M. L. Begeman. gIBIS: A hypertext tool for exploratory policy discussion. *Proc. of CSCW '98*, p. 140–152.
2. J. Donath, K. Karahalios, and F. Viegas. Visualizing Conversations. *Proc. of HICSS 32*, Jan 1999.
3. S. Farnham, H. Chesley, D. McGhee, R. Kawal, J. Landau. Structured Online Interactions: Improving the Decision-Making of Small Discussion Groups. *Proc. of CSCW 2000*, p. 299–308.
4. F. Flores, M. Graves, B. Hartfield, T. Winograd. Computer Systems and the Design of Organizational Interaction. *ACM Trans. on Information Systems* 6(2), p. 153–172, 1988.
5. J. Grudin. Groupware and Social Dynamics: Eight Challenges for Developers. *Comm. of the ACM* 37(1), p. 92–105, Jan. 1994.
6. G. C. Hill, R. King. Dry Spell: A Startup Survival Guide. *Business 2.0*, Sep. 2000.
7. W. Kunz, H. Rittel. Issues as elements of information systems. Working Paper 131, Institute of Urban and Regional Development, University of California, Berkeley, 1970.
8. S. Rohall, D. Gruen, P. Moody, S. Kellerman. E-mail Visualizations to Aid Communications. *Proc. of IEEE Symposium on Information Visualization 2001*.
9. M. Smith, A. T. Fiore. Visualization Components for Persistent Conversations. *Proc. of CHI 2001*, p. 136–143.
10. M. Smith, J. J. Cadiz, B. Burkhalter. Conversation Trees and Threaded Chat. *Proc. of CSCW 2000*, p. 97–105.