Proceedings of the Third European Conference on Computer-Supported Cooperative Work
13-17 September, 1993, Milan, Italy
G De Michelis, C. Simone and K. Schmidt (Editors)

Low overhead, loosely coupled communication channels in collaboration

Dorab Patel and Scott D. Kalter
Twin Sun, Inc., USA

Abstract: Communication and coupling are two central aspects of systems developed for computer-supported cooperative work. Synchronous communication usually implies tight coupling while asynchronous communication is often used with loose coupling. This paper explores the previously neglected role of loosely coupled channels in synchronous communication by providing some example channels and evaluating their tradeoffs.

Such loosely coupled channels efficiently meet specialized communication needs that often arise in spontaneous, short-lived collaborations. They can also augment existing channels in specific domains.

These channels impose few requirements on their host applications and hence can be easily integrated into tools familiar to most users. Our implementation is built over an inter-application communication framework that provides flexible high-level communication abstractions for the rapid prototyping, implementing, and experimenting with these channels.

1 Introduction

There is increasing interest in developing computer-based tools to promote collaboration. One important aspect of collaboration is the communication of information. This communication may be synchronous (all participants are present at the same time) or asynchronous (all participants need not be present simultaneously). In both of these cases, the coupling between the participants can be loose, tight, or intermediate. Tight coupling, as in shared whiteboards, involves each participant knowing continuously about changes made by the others. The application state may be distributed, but all the copies are kept consistent continuously. Loose coupling means that the participants use independent applications which use one-shot, application-initiated information exchanges to copy state among one another. No attempt is made to keep the copies consistent after the state transfer. Previously, the emphasis in synchronous collaborative systems has been on tight coupling. This

paper explores the role of loosely coupled communication channels in synchronous collaboration.

One can imagine a world without single-user applications. All applications would be tightly coupled and group aware. Users would not need to do anything different to share their work with others. But, we do not live in such a world. Existing tightly coupled systems usually incur a cognitive overhead for collaboration. By "cognitive overhead" we mean the extra mental effort required to set up a collaboration and to establish a shared context, over and above the actual collaboration. Also, tightly coupled systems are often too slow or too expensive to implement. Since most popular applications today are written with a single-user mind-set, it is difficult to convert them to multi-user use. Hence many group-aware applications are unfamiliar to users who are reluctant to give up their comfortable single-user tools. This tendency retards the spread of collaborative applications.

In a tightly coupled system, users have to be aware of others. They do not have the flexibility of decoupled use and might inadvertently do something with adverse effects on the others. Also, users might be inhibited if everything they do is visible to others. These problems can be solved by having a separate private data space that can be used in these circumstances. The data location, rather than the multi-user application, would control the data sharability.

Though a tightly coupled system is useful for situations that require a continuous and highly coherent sharing of context (e.g., meetings), there are other situations for which it is not as appropriate. Human collaborations are often spontaneous, unstructured, and have short lifetimes. For example, you ask someone a quick question; or you solicit an opinion on some of your work. A loosely coupled communication channel accurately models this situation and hence is familiar to users. This type of collaboration only works if setup and shutdown are a small fraction of the total collaborative session. A low cognitive overhead for collaboration makes it more likely that the system will be used for short-lived spontaneous collaborations. A low implementation overhead makes it easier to insert these features into existing applications, increasing the likelihood of making such systems widely available. By implementing shared multi-writer environments, a tightly coupled system provides facilities that are beyond those normally available. The availability of such features is one reason why tightly coupled systems have received more attention. However, a loosely coupled channel with limited benefits may still be desirable if its costs are sufficiently low.

To use an analogy with network protocols, loosely coupled communication channels are similar to datagram communication, and tightly coupled channels are similar to virtual circuits. Like loosely coupled channels, datagrams are suited to short, low overhead communications. Virtual circuits, like tightly coupled channels, require the overhead of setting up and tearing down a connection, but provide efficient long term communications.

Channels at different levels of coupling will be suited to different collaborative situations. The benefit provided by a particular channel has to be evaluated with respect to its implementation and cognitive costs. Some provide lots of functionality

Type		Examples
Video		VIDEODRAW, VIDEOWINDOW, CLEARBOARD
Asynchronous	loosely coupled	Email, USENET
	annotations	PREP, FOR COMMENT
Synchronous	application sharers	SHAREDX, XTV, SHOWME
	toolkits	GROUPKIT, SUITE, RENDEZVOUS, DISTEDIT, COEX
	tightly coupled	WSCRAWL, GROUPDESIGN, SHREDIT, ASPECTS

Table 1. Existing channels

at a high cost. Others provide less functionality but at a much reduced cost. Choosing the right one depends on the application. We focus on loosely coupled channels that incur minimal costs and have proven to be useful in our environment.

This paper begins with a survey of existing communication channels for collaboration, followed by examples of new, low cost, and useful channels. A taxonomy of communication channels provides a basis for comparison of various alternatives. We touch on our implementation environment which allows us to experiment with new channels easily, and application functionality necessary to permit integration of such channels. We conclude with lessons learned in building and using low cost loosely coupled communication channels for synchronous collaboration.

2 Existing communication channels

Before collaborating, participants must develop a shared context. This context is created by the exchange of information across communication channels. Each channel has its own characteristics and affordances (Gaver, 1992). Channels may or may not be computer-supported. Channels such as paper, whiteboards, phone, fax, and video are usually not well integrated with the workstation. Paper and whiteboards are convenient, ubiquitous, and familiar, but require co-location of participants. Phone, fax and video require more session setup but can be used remotely. The rest of the paper concentrates on computer-supported channels.

Computer-supported channels can support and augment collaborations. The more channels that are available to the participants, the better, so they can choose the channel appropriate to the task at hand. We survey some existing channels, provide examples of each, and review their characteristics.

Video channels (e.g., VIDEODRAW (Tang & Minneman, 1990), CLEARBOARD (Ishii, Kobayashi & Grudin, 1992), and VIDEOWINDOW (Fish, Kraut & Chalfonte, 1990)) allow free-form interactions among participants. These

systems use data that is external to the computer, though CLEARBOARD-2 includes a sketching tool called TEAMPAINT. This lack of integration limits the ability to manipulate the data and eliminates the possibility of maintaining semantic relationships with other stored computer data.

Asynchronous, loosely coupled channels such as email and USENET are passive systems, limited to a single writer for each document. However, they allow participants to communicate when convenient, rather than being present simultaneously in order to communicate.

Asynchronous annotation systems like PREP (Neuwirth, Kaufer, Chandhok & Morris, 1990) and FOR COMMENT allow participants to work at different times, and allow multiple writers to annotate the same document. PREP even provides a computer-maintained relationship from the annotations to the underlying document.

Synchronous application multiplexers such as SHOWME, XTV (Abdel-Wahab & Feit, 1991), and SHAREDX (Garfinkel, Gust, Lemon & Lowder, 1989), allow any application to be shared in a strict WYSIWIS (Stefik, Bobrow, Foster, Lanning & Tatar, 1987) manner. This can be beneficial in certain situations such as teaching or presentations. However, these applications require premeditation—users must know, before starting an application, whether they are going to be sharing it or not. Strict WYSIWIS can be a hindrance in situations where participants require independent views of shared data. SHOWME shares dynamic annotations over a static snapshot of an application. The others allow continuous sharing of the underlying application.

Toolkits (e.g., RENDEZVOUS (Hill, Brinck, Patterson, Rohall & Wilner, 1993), GROUPKIT (Roseman & Greenberg, 1992), COEX (Patel & Kalter, 1993), DISTEDIT (Knister & Prakash, 1990), and SUITE (Dewan & Choudhary, 1991)) for developing tightly coupled applications that share their data among application instances, have been developed recently. They provide fine grained sharing among participants, but require modification of existing applications. RENDEZVOUS uses a centralized state architecture, whereas all the others use a decentralized state approach.

Tightly coupled synchronous applications like SHREDIT (McGuffin & Olson, 1992), WSCRAWL (Wilson, 1992), ASPECTS (Group Technologies, Inc., 1990), and GROUPDESIGN (Beaudouin-Lafon & Karsenty, 1992), share the characteristics of the toolkits mentioned previously. Since these applications are new, they are unfamiliar to users who resist converting to using these applications. WSCRAWL uses a centralized state approach. All the others use a distributed state approach.

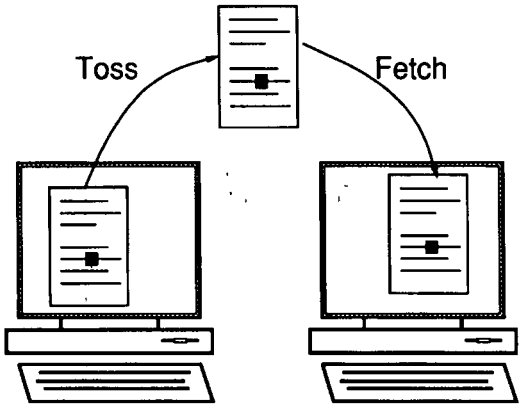

Figure 1. TOSS/FETCH

3 New communication channels

This section introduces some new computer-mediated communication channels. These channels are similar to some common, existing channels and thus are familiar. Each channel is designed to be unobtrusive to the communicators, so they can concentrate on the substance rather than the method of their collaboration. The primary goal of a new channel is either to enrich an existing communication channel, or to replace an existing channel with one that is less cumbersome. These channels were motivated by our own daily needs in designing systems, developing code, and writing papers. Our work style is such that we often work closely together, often at the same desk, and were hampered by the lack of collaborative tools.

3.1 Toss/fetch

A common need is to show a colleague what text file we are examining, from our own point of view. This is usually to establish a context for further discussion. Typically, we would walk into their office or call them and ask them to look at the relevant part of a file. The problem with this is that the conversation may deviate from the substance of the discussion to describing how to establish the context. For example, the file might be accessed via a different path on the colleague's host, it might be read protected, or locating the relevant region of the text might be tedious.

One solution would be to take a snapshot of our window and display it on their screen,[1] but this results in a static view. Another possibility is to use either SHAREDX or XTV. These are rather heavyweight solutions that require premeditation of the need to share, and different modes of use while sharing (floor-passing and session management).

A lightweight system, that provides one user's point of view (POV) within an application to another user within their own application context, is more appropriate. The received view is read-only, thereby avoiding complex consistency issues,

[1] A trivial implementation using X: `xwd | xwud -display host:display`.

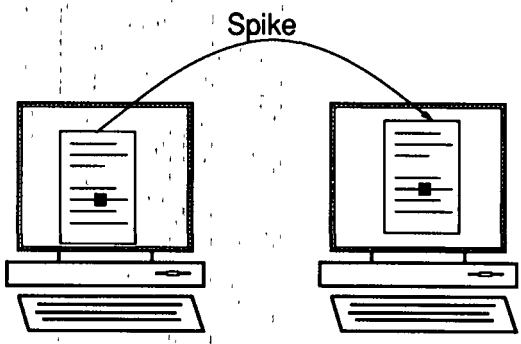

Figure 2. SPIKE

keeping the channel loosely coupled, and its implementation simple. This provides recipients with a dynamic, rather than static, view—one they can scroll or otherwise manipulate independently. Furthermore, although they cannot modify the original document, they can modify their copy normally.

Given these constraints, we developed a channel called TOSS/FETCH (see Figure 1). One user can toss the point of view within their own application and another user can then fetch this POV. The POV consists of the file being viewed, the cursor location, and, if the editor supports it, where the "mark" is. Our experience indicates that this definition of a POV includes sufficient information to establish a context for collaboration involving text editors..

There are many benefits to the TOSS/FETCH mechanism. Since the tossed POVs are globally visible, the communication is undirected, as in USENET. The undirected communication leads to a very simple user interface. It requires only one keystroke to toss or fetch a POV. There is no automatic notification associated with this channel. The sender is expected to inform the receiver, through other channels, that there is something to fetch. Typically, the participants will already be conversing at this point, so informing the receiver is not a problem. The modest requirements of the TOSS/FETCH mechanism impose very few demands on the host application. Hence it is easy to implement TOSS/FETCH in familiar applications preferred by users. To demonstrate the ease of implementation, we implemented TOSS/FETCH for EMACS (Stallman, 1987) (no C changes, only ELISP (Lewis, LaLiberte & the GNU Manual Group, 1990) additions), VI (no changes, addition of two macros), and STED (less than 200 lines of C++ added; all but 4 in a new file). The TOSS/FETCH concept can be extended to different media (e.g., video) or differently structured information (e.g., hypertext).

The channels described next provide added functionality at the cost of imposing greater demands on the application.

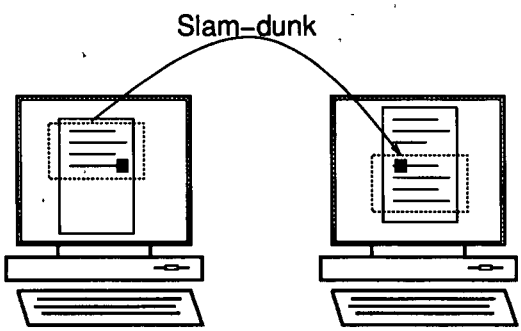

Figure 3. SLAM-DUNK

3.2 Spike

There are a few problems with TOSS/FETCH.

- The recipient has to be notified about a toss via an alternate channel.
- The tossed POV is visible to all users and hence may not be suitable for private communications.
- The recipient has to take action to fetch a POV.
- Since a fetch retrieves the last tossed POV, if multiple users are tossing and fetching simultaneously, they may interfere with each other.

SPIKE (see Figure 2) was developed to address these problems. SPIKE is a directed communication channel for transmitting POVs directly to the intended user. SPIKE requires the ability of applications to exchange messages, leading to a more complex implementation than for TOSS/FETCH. To spike a POV the sender activates SPIKE with a single keystroke and then specifies the recipients. The recipients of the SPIKE will be presented with the POV on their display.

The sender's cognitive overhead is increased since a target must now be specified. However, the recipient's overhead is reduced since the POV is directly deposited into a distinguished buffer of their application. The notification is now active and the recipient does not need to take any action to retrieve a POV. Since the communication is directed, multiple users can use this channel without interfering with each other, and only the intended recipient sees the POV.

3.3 Slam-dunk

Sometimes there is a need to take a text fragment from one user's buffer and insert it into the buffer of another. This situation occurs often when two people work together in close proximity, but only one of them is directly editing the object of collaboration. The other writes some text that must be transferred into the buffer of the first.

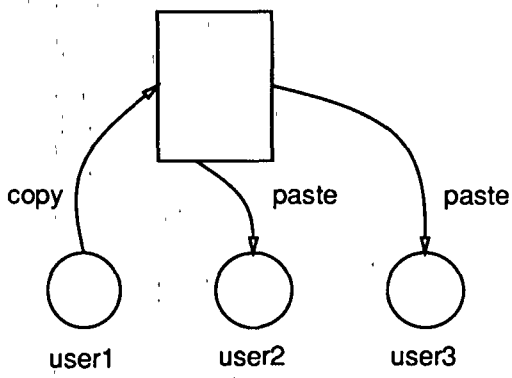

Figure 4. A global clipboard

One way of transferring the data is to write out the text fragment into a file and have the primary author read the file in at the appropriate location. This approach suffers from the same security and path name problems mentioned earlier. Alternatively, email, TOSS/FETCH, or SPIKE could be used to transfer the fragment. The recipient must then explicitly copy the fragment over to the desired location.

We developed a channel, SLAM-DUNK (see Figure 3), that allows the sender to select a region of text and send it to a specified user. The recipient's application automatically places the received text at the current location of the cursor. SLAM-DUNK requires some external coordination to prepare the recipient to receive the fragment. This coordination could be eliminated by complicating the user interface. The recipient could simply be notified that data had arrived in a distinguished buffer. The data could then be inserted wherever necessary with a single keystroke. Our experience is that leaving the coordination outside the system seems quite natural in this situation.

3.4 Clipboards

A multi-user clipboard is a reasonable generalization of the previous channels. However, this generality can make the clipboard more cumbersome to use. Where the data is stored and how it is accessed defines the clipboard design space. We assume that a single clipboard contains a single object and manages cut, copy, and paste operations.

3.4.1 Global clipboard

In the simplest case there is one global clipboard that everyone can copy to and paste from (see Figure 4). This is similar to TOSS/FETCH since it is undirected. The advantage to this clipboard is that the interface is exactly like that of a standard clipboard, making it familiar. However, a potent disadvantage is that the probability of interference is high. Many users copying to the same clipboard will overwrite each others' information. This may be acceptable in a small user community, but

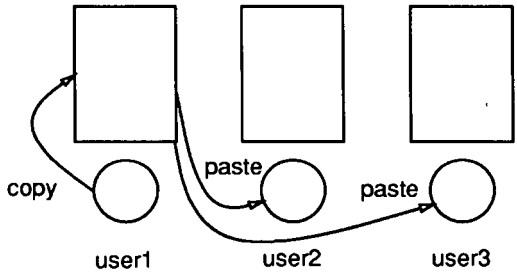

Figure 5. A public clipboard

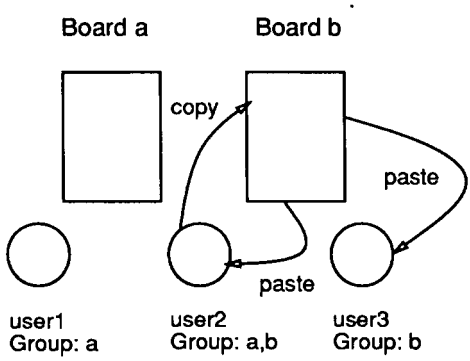

Figure 6. Group clipboards

not in a large one.

3.4.2 Public clipboard

A variation is to make a separate public clipboard (see Figure 5) for each user that is stored in the user's own application memory. Operating on a user's own clipboard is the default. A user also can copy to or paste from another's public clipboard. This directed communication greatly reduces the problem of interference as the user community grows, at the cost of only a minor complication of the interface. However, the clipboard has to be public so that another user can cut to or paste from one's clipboard. Adding security would alleviate this problem at the expense of unduly complicating the interface.

3.4.3 Group clipboard

An intermediate variation between the global clipboard and public clipboards is to have a clipboard for every group (see Figure 6). Only members of a particular group would be able to access the group's clipboard, thus reducing privacy concerns. Group clipboards also alleviate the interference problems caused by a large user community. The overhead of specifying the target of a cut or paste can be reduced by choosing the last target as the default.

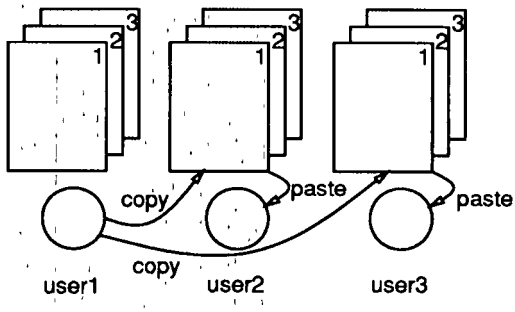

Figure 7. Private clipboards

3.4.4 Private clipboard

The final variation is to provide each user with a separate, private clipboard (see Figure 7) to communicate with other users. This provides each pair of users with two distinct privately shared clipboards (a local one and a remote one). Each user also has a private clipboard for their own use. This scheme provides flexibility in communication and security, at the cost of making the model and interface more complicated. The target clipboard is normally specified by a user identifier. Depending on the design of the system, additional information may be necessary to disambiguate between the user's local and remote clipboards. As the communication is directed, no special security mechanisms are necessary. For two users to communicate information with this channel, two directed operations must be used. The first user copies explicitly to the second user who then pastes from the first.

The channels that we have described are just examples. Other variations are conceivable. These channels are important because they are simple and they either enrich existing communication channels, or make some forms of communication less cumbersome, or both. Their simplicity makes it easy to understand what they will do under different circumstances and allows for easy integration into a system or application design. We have found them to be reasonably useful and powerful in our own collaborative tasks.

4 A taxonomy of communication channels

This section provides a taxonomy of various communication channels based on selected attributes. The cognitive overhead of each attribute highlights the tradeoffs in choosing a particular channel over another for a given situation. A channel involves communication from a sender to one or more receivers. The attributes listed below are in chronological order of occurrence during a communication. Not all the attributes are orthogonal to each other. Table 2 summarizes the channels described in this paper. Synchronous applications are included to provide a point of comparison.

	TOSS/FETCH	SPIKE	SLAM-DUNK	clipboards	sync apps
User interface	snapshot	snapshot	snapshot	snapshot	continuous
Source data	both	both	region	region	both
Target	undirected	directed	directed	either	directed
Recipient	broadcast	unicast	unicast	unicast	multicast
Transport	async	sync	sync	either	sync
Notification	passive	active	active	both	active
Destination data	buffer	buffer	point	point	NA
Recipient action	yes	no	no	yes	no

Table 2. Channel taxonomy

4.1 Interface type

What kind of interface does the channel provide to the sender? Is it similar to a datagram, involving minimal specification of the collaborative session; or is it similar to a virtual circuit, which involves setting up the channel?

In a datagram-like situation, the communication is either undirected or uses easily specifiable addresses with good use of defaults. Thus, there is little cognitive overhead for a sender. The low overhead makes this type of channel appropriate for quick, short-lived communications. Examples include TOSS/FETCH, SLAM-DUNK, and SPIKE.

Longer communications can benefit from expending effort in setting up a session, since there is an expectation of a larger number of longer communication events. The user has to set up the application and collaboration participants before the collaboration can occur. Examples include XTV, WSCRAWL, and SHOWME.

4.2 Source data

What does the source data include? The data sent as part of the communication may be a whole buffer, a region in a buffer, or both. In the case of a buffer, the sender requires less specification since the buffer can be assumed to be the one containing the current location by default. Sending a region at least involves specifying its extent. SPIKE uses a whole buffer, SLAM-DUNK uses a region, and TOSS/FETCH uses both a buffer and a region.

4.3 Targets

Is the target of the communication directed or undirected? Directed communication involves a greater cognitive overhead than undirected communication because the targets require extra specification. This overhead may be reduced by the use of good default schemes. This attribute has some overlap with the interface type attribute. For example, email is usually directed, whereas USENET is relatively undirected communication.

4.4 Recipients

Can multiple recipients receive the communications? Some communications are point-to-point or unicast; others are multicast or broadcast. This attribute is similar to the previous one except that the target is from the sender's point of view whereas the recipient is from the receiver's point of view. Unicast requires more specification than broadcast and hence more overhead. TOSS/FETCH is an example of broadcast, whereas SPIKE exemplifies unicast.

4.5 Transport

Is the transport asynchronous or synchronous? This attribute does not have direct impact on user overhead, except that asynchronous transports usually imply loosely coupled systems like email, while synchronous transports are usually used for tightly coupled systems like XTV.

4.6 Notification

Is the notification active or passive? On the receiving side of the communication, the receipt of a message can either be actively notified to the recipient or have the recipient check for the message. Passive notification puts a greater burden on the user. SLAM-DUNK can be considered to have active notification, while USENET typically has passive notification.

4.7 Destination data

Where does the received data go? Most often, the destination of the communicated data is a new or existing buffer or file. However, as in the case of SLAM-DUNK, the destination may be a location in an existing buffer. The latter incurs an overhead for the recipient since the cursor has to be positioned at the correct location.

4.8 Recipient action

Does the recipient have to take any action on the receipt of a communication? This is similar to notification, but focuses on the actions the recipient takes after notification. For example, email usually has active notification, but requires the receiving user to read in the new message using a mail user agent. On the other hand, SPIKE requires no recipient action since the message is automatically delivered into a new buffer.

5 Implementing communication channels

Most of the channel prototypes were built in a environment that we developed to experiment with communicating applications and to explore both loosely coupled and tightly coupled groupware. The environment allows EMACS processes to communicate data between one another. We use an EMACS process as a server and other EMACS processes as clients which led to the name ESC (Emacs Server Client).

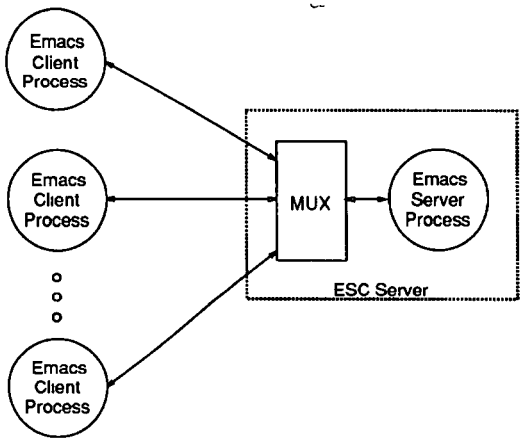

Figure 8. ESC Process Architecture

We implemented the environment in EMACS using an object-oriented extension to ELISP called EOOPS (Houser & Kalter, 1992).

The ESC server is actually two processes, an EMACS running ESC server code and an additional process called the ESC-mux which is used to listen on ports and do non-blocking writes (see Figure 8). Performing these operations in a single EMACS process would have required low-level modifications to EMACS, which we wanted to avoid.

ESC provides reusable classes that solve many low level problems in communication, resource identification, and session management so that experimenters can concentrate on higher level issues. The class library provides support for creating and maintaining connections with the server, managing client registration information, sending and receiving messages, notifying applications of events, and writing user interfaces. ESC is not meant for production systems but to provide extremely flexible and high level primitives with which to experiment. It also provides a basis with which to modify these primitives to seek better abstractions, and to improve functionality and performance.

Implementing channels in this environment limits their availability to applications written in the ESC framework. The advantage of ESC is that it provides high level primitives to prototype and experiment with working versions of an experimental channel quickly. For instance, with ESC support, the classes that implement SPIKE and SLAM-DUNK are about 50 lines of EOOPS code each. The first version of SLAM-DUNK was completed within a couple of hours of recognizing its potential usefulness.

These mechanisms could be implemented using USENET or email as a transport mechanism. This would certainly be possible, but a notification mechanism, API, and user-interface would also have to be developed in order to integrate the communication into actual applications.

6 Necessary application characteristics

A significant benefit of loosely coupled communication channels is that they require minimal support from the application. Hence it is easy to implement such channels for almost any application. This means that users can continue to use familiar applications, increasing the likelihood that these channels will be used. These requirements are listed below.

Command loop modifications Most interactive applications have a command or event loop. It is usually trivial to add a new command to such applications.

Read/write document fragments The ability to read data from a section of the document and write to a section of the document without disturbing the rest of the application state is required.

Get/set the current point and mark The ability to retrieve and set the current cursor location is required. If the application supports the notion of a mark, retrieving and setting the mark is useful too.

Communication support Low-level communication system support is required for ESC-supported applications, like SPIKE and SLAM-DUNK, which do not require recipient intervention. The communication is required for active notification so that the application can perform some action upon certain events without user interaction.

Fortunately, most interactive applications support the first three requirements with very little effort. Event-driven applications also support the last requirement trivially, by allowing additional communication channels to be hooked into their event loop. The low-level communication facility is important. ESC provides this for EMACS. Other applications can use other inter-application communication frameworks such as TOOLTALK (SunSoft, 1991), ISIS (Birman & Joseph, 1986), COEX (Patel & Kalter, 1993), and SOFTBENCH (Cagan, 1990).

7 Future work

The communication channels mentioned earlier are examples of loosely-coupled channels that are useful during collaboration. Many more variations on this theme need to be tried out to determine their value in practice. We have used these communication channels within our group and found them to be useful. Systematic external user testing of promising channels is required before the usability claims can have more general validity. Our ESC development environment was instrumental in allowing us to explore alternatives rapidly. As a result, most of our experimentation has used EMACS. Adapting these channels for use in other applications will help evaluate the applicability of this approach to other domains.

8 Conclusion

Our experience, as reported in this paper, has shown that loosely coupled communication channels are easy to implement and are useful in providing low-overhead channels for meeting specific communication requirements during a collaboration. Existing applications can integrate these channels with ease since the application requirements are moderate.

Low cost, loosely coupled channels for specialized communication, or for augmenting existing channels, are useful in limited domains. The ability to implement and experiment with such a channel quickly is important. Inter-application communication frameworks, like ESC, provide flexible, high-level communication abstractions that greatly facilitate rapid experimentation with various loosely coupled channels.

References

Abdel-Wahab, H. M. & Feit, M. A. (1991). "XTV: A framework for sharing X window clients in remote synchronous collaboration". In *Proceedings of the IEEE TriComm'91: Communications for Distributed Applications and Systems*, April 1991, pp. 159–167, Chapel Hill, North Carolina.

Beaudouin-Lafon, M. & Karsenty, A. (1992). "Transparency and awareness in a real-time groupware system". In *Proceedings of the ACM Symposium on User Interface Software and Technology UIST'92*, November 1992, pp. 171–180, Monterey, CA. ACM Press.

Birman, K. P. & Joseph, T. A. (1986). "Communication support for reliable distributed computing". Technical Report TR 86-753, Computer Science Department, Cornell University, Ithaca, NY.

Cagan, M. R. (1990). "The HP SoftBench environment: An architecture for a new generation of software". *Hewlett-Packard Journal*, *41*(3), 36–47.

Dewan, P. & Choudhary, R. (1991). "Primitives for programming multi-user interfaces". In *Proceedings of the ACM Symposium on User Interface Software and Technology UIST'91*, November 11–13 1991, pp. 69–78, Hilton Head, South Carolina. ACM Press.

Fish, R. S., Kraut, R. E., & Chalfonte, B. L. (1990). "The VideoWindow System in informal communication". In Halasz, F. (Ed.), *CSCW90: Proceedings of the Conference on Computer-Supported Cooperative Work*, October 7–10 1990, pp. 1–11, Los Angeles, California.

Garfinkel, D., Gust, P., Lemon, M., & Lowder, S. (1989). "The SharedX multi-user interface user's guide, version 2.0". Technical Report STL-TM-89-07, Hewlett Packard Laboratories, Palo Alto, California.

Gaver, W. W. (1992). "The affordances of media spaces for collaboration". In Turner, J. & Kraut, R. (Eds.), *CSCW92: Proceedings of the Conference on Computer-Supported Cooperative Work*, October 31–November 4 1992, pp. 17–24, Toronto, Canada.

Group Technologies, Inc. (1990). *Aspects User Manual*. Arlington, Virginia: Group Technologies, Inc.

Hill, R. D., Brinck, T., Patterson, J. F., Rohall, S. L., & Wilner, W. T. (1993). "The Rendezvous language and architecture". *Communications of the ACM*, *36*(1), 62–67.

Houser, C. & Kalter, S. D. (1992). "Eoops: An object-oriented programming system for Emacs-Lisp". *LISP Pointers*, *5*(3), 25–33.

Ishii, H., Kobayashi, M., & Grudin, J. (1992). "Integration of inter-personal space and shared workspace: ClearBoard design and experiments". In Turner, J. & Kraut, R. (Eds.), *CSCW92: Proceedings of the Conference on Computer-Supported Cooperative Work*, October 31–November 4 1992, pp. 33–42, Toronto, Canada.

Knister, M. J. & Prakash, A. (1990). "DistEdit: A distributed toolkit for supporting multiple group editors". In Halasz, F. (Ed.), *CSCW90: Proceedings of the Conference on Computer-Supported Cooperative Work*, October 7–10 1990, pp. 343–355, Los Angeles, California.

Lewis, B., LaLiberte, D., & the GNU Manual Group (1990). *The GNU Emacs Lisp Reference Manual* (1.02 ed.). Cambridge, Massachusetts: Free Software Foundation.

McGuffin, L. & Olson, G. M. (1992). "ShrEdit: A shared electronic workspace". CSMIL Technical Report 45, University of Michigan, Ann Arbor, Michigan.

Neuwirth, C. M., Kaufer, D. S., Chandhok, R., & Morris, J. H. (1990). "Issues in the design of computer support for co-authoring and commenting". In Halasz, F. (Ed.), *CSCW90 · Proceedings of the Conference on Computer-Supported Cooperative Work*, October 7–10 1990, pp. 183–195, Los Angeles, California.

Patel, D. & Kalter, S. D. (1993). "A UNIX toolkit for distributed synchronous collaborative applications". *Computing Systems*, *6*(2). To appear.

Roseman, M. & Greenberg, S. (1992). "GroupKit: A groupware toolkit for building real-time conferencing applications". In Turner, J. & Kraut, R. (Eds.), *CSCW92: Proceedings of the Conference on Computer-Supported Cooperative Work*, October 31–November 4 1992, pp. 43–50, Toronto, Canada.

Stallman, R. (1987). *GNU Emacs Manual* (Sixth ed.). Cambridge, Massachusetts: Free Software Foundation.

Stefik, M., Bobrow, D. G., Foster, G., Lanning, S., & Tatar, D. (1987). "WYSIWIS revised: Early experiences with multiuser interfaces". *ACM Transactions on Office Information Systems*, *5*(2), 147–167.

SunSoft (1991). *ToolTalk 1.0 Programmer's Guide*. Mountain View, California: SunSoft.

Tang, J. C. & Minneman, S. L. (1990). "VideoDraw: A video interface for collaborative drawing". In Chew, J. C. & Whiteside, J. (Eds.), *CHI'90: Proceedings of the Conference on Human Factors in Computing Systems*, April 1–5 1990, pp. 313–320, Seattle, Washington. ACM, Addison Wesley.

Wilson, B. (1992). "WSCRAWL 2.0: A shared whiteboard based on X-Windows". Technical Report 33, Apple Computer Library, Cupertino, California.