Strudel - An Extensible Electronic Conversation Toolkit

Allan Shepherd, Niels Mayer, Allan Kuchinsky

Hewlett-Packard Laboratories
1501 Page Mill Rd
Palo Alto, California 94304.
shepherd@hplabs.hpl.hp.com

ABSTRACT

This paper describes the conceptual model of *Strudel*, a toolkit of generic components for conversation and action management. To empower work groups to more effectively conduct their computer-based communication, coordination, and information sharing activities, *Strudel* packages within a simple model of task and action the semi-structured message, active message and conversation management paradigms. To facilitate acceptance and use within varying work cultures, we define this model in terms of a set of extensible components, which are implemented as a prototype software toolkit that is efficient, portable, customizable, and extensible. Issues considered briefly in this paper include threading in conversations that are converging or multi-party, and interoperability between active message systems.

INTRODUCTION

Our project is investigating the potential for achieving better management of computer-based conversations in work groups through technologies that enable teams to capture and structure discourse. We believe this to be useful for work groups engaged in specialized, recurring conversations, where building specific conversational structures for certain tasks allows them to more effectively coordinate their activities when carrying out these tasks. Examples of such specialized conversations include coordination of software engineering activities and deliberation of system design issues.

Many work groups within Hewlett-Packard routinely use computer-based conversations to deliberate on design decisions [Fafc90], to track and follow up on negotiations and agreements, and to schedule their activities. These work groups have evolved ad hoc structures and conventions to carry out specialized conversations which include the use of specialized mail templates and incorporation of the mail system within applications, e.g. to send notification messages in a software defect tracking system.

To enable users to more easily integrate the usage of computer-based conversations into other electronically supported work, we are exploring more systematic mechanisms for structuring computer-based conversations. The current focus of *Strudel* is on how users can interactively carry the state of each conversation or task forward, rather than on automated response to events; for example when a user replies to a message, or marks an action item in a task as "Done."

Overview of the Technical Approach

Strudel provides a toolkit of components for end-users to manage electronic mail (e-mail) based conversations and action items. This paper describes the design of the toolkit components, with examples from the current prototype. To facilitate acceptance, the toolkit emphasizes standards, user-extensibility of toolkit components, interoperability

with existing applications, and good run-time performance on widely installed platforms. It is compatible with existing practice, for example it has a similar look and feel to existing e-mail user agents. It is aimed at producing a small, fast and portable C-implemented platform that enables delivery of groupware applications on relatively low-cost graphics workstations running industry standard software — UNIX,[1] the X Window System (Version 11) [Sche88] and ARPA Internet mail [Post82]. User-acceptance of *Strudel* is further addressed through its use of the graphical interface provided by the OSF/Motif UI Toolkit [Moti90]. To support tailoring, *Strudel* provides an extension language which is based on *Winterp* [Maye90].[2] An early experimental prototype of *Strudel* was demonstrated at the IFIP Groupware Technology Workshop [Shep89].

As a starting point, we have integrated the main features of other conversation management systems into a simple conceptual model. This builds upon work on e-mail user-agents that support message filtering [Rose86, Bore88], and work on semi-structured message systems [Malo86], and conversation management [Wino87, Come86, Doll89, Sulo90, Kapl90]. *Strudel* contains a library of components which include user-customizable definitions for conversations, conversational moves, actions, action items and notifications. Presenters allow viewing, editing and navigating among these objects.

Previous work [Malo86] has focussed on making the flood of incoming electronic mail manageable through support for the rules that filter messages into message classes for presentation in browsers,[3] and for actions that can be applied to messages in a class, such as "forward all messages from X to Y." *Strudel* complements this work by focussing on facilities that will help users (within groups) add partial structure and actions to messages while messages are being composed. *Strudel* supports this addition of structure to messages during composition by providing a library facility in which user-extensible types for message and conversation components are managed.

In *Strudel*, messages can contain typed *conversational moves*, such as a "Request," as in Conversations for Action [Wino87]. We use the name conversational move, rather than message "type," since messages are allowed to contain more than one such "move," and to emphasize that each move may suggest next moves that are typically taken by other parties in a conversation. A move may have *fields*, as in semi-structured messages. For example, a "Request Meeting" move may have date, location, and topic fields. A move may contain *actions*, for example, the "Request Meeting" move may contain an "Add to To-Do List" action. which may access the move's fields. Users draft messages by selecting a move from a top-level menu, or via buttons, menus or lists of items displayed in previous messages.

Messages are collected into conversations based on the threading between successive conversational moves. Conventional electronic mail messages and *Strudel* messages may be freely inter-threaded within conversations. Conventional electronic mail messages are traced to a predecessor message using existing "In-reply-to:" or "Subject:" field entries

---

[1]UNIX is a trademark of AT&T.

[2]*Winterp* was released on the X11r4 tape — it provides an interactive object-oriented interface to the OSF/Motif UI Toolkit, using XLisp's light-weight interpreter and object system.

[3]This filtering is based either on explicit message typing, or some pattern matching with the content of message fields. In semi-structured message systems, the sub-structure in the message body is used to simplify how the user defines predicate matching. It is also used to allow actions on the messages to interpret particular fields. The utility of these systems depends on their ability to classify incoming messages and on the actions that can be applied to these classes of messages.

where possible. *Pseudo-conversations* [Come86], i.e. collections of messages with the same topic, or sender, etc, will be supported. Actions can be applied to messages classified into pseudo-conversations, for example, an action "Request an item from library" may be defined for a class of messages that contain lists of new library aquisitions.

Strudel differs from previous approaches in that users can dynamically evolve conversational move and conversation type definitions. Therefore these definitions can be made task specific. For example a user may create a new conversational move type "Ask who is responsible for repairing a medical instrument defect." The user may then add this move as an initial move type in a "Medical Instrument Defect Repair" conversation type. However there is currently little support for integrating new definitions of moves or conversations into a centralized library, as would be necessary to support COSMOS style scripts. Unlike more complex office procedure and task modelling systems [Deci86, Ishii89, Krei89], *Strudel* supports only simple scripts to help end-users select and draft next moves in conversations and tasks.[4]

The actions supported on messages and other objects can also be made task specific. An interface to other tools is supported so that actions specified within conversation and task objects can initiate operations on objects managed by other tools. In particular, simple programmatic interfaces to general purpose applications such as a room reservation system, and to domain specific coordination tools such as software maintenance and defect tracking systems, can be defined. For example, a "Request meeting" message may have an action "Make room reservation" — this action could invoke an operation in a room reservation application.

To allow users to tie the message system into the state of their tasks, users can create *action items* in Strudel. Action items are memos created by a user to describe an activity they intend to carry out, and the status of that activity. For example, a software development engineer may create an action item to note an intended defect repair, its priority, etc. In addition, to inform *Strudel* users of an event in an external tool, a special kind of action item, named a *notification*, can be created by a call from the tool. Analogous to messages in conversations, action items are represented as semi-structured typed forms and threaded into *tasks*. *Action-item* forms are distinct from *actions*.

The evolution of message and conversation types is decentralized and done by individual users, but is expected to be mediated through a group's discussion and acceptance of modified types. Thus as groups adopt methodology or protocols for their work process, they may choose to represent some conversation and task activities in *Strudel*. *Strudel* does not advocate particular protocols but rather tries to provide ways for groups to support their protocols of choice, and to allow groups to informally integrate and then specialize these.

Strudel conversation types do not restrict the types of next moves that can be made; therefore different conversation types can be freely initiated at any point in a conversation. For example, a "Request meeting" or a "Post design issue" move can be sent in response to a "Defect notification" received in the defect resolution conversation.

Example of Usage

Detailed application scenarios have been developed with potential users. To support the scenarios prototyped so far, several *conversation types* have been defined in the library.

---

[4]Expert users can define complex task actions in *Strudel* by writing interpreted procedures in *Winterp* [Maye90].

These include a conversational IBIS[5] used for design issues discussion, a Conversation for Action [Wino87], meeting scheduling, and software defect tracking and repair.

Throughout the paper, a defect resolution scenario is used to give examples. In this scenario, a software build system notifies *Strudel* that a defect in a product assembly was found. A software technician responsible for the build process uses *Strudel* to read pending notification reports. The defect notification is presented as a graphical form. Buttons on the form allow the software technician to draft and send specific types of messages, for example, a message to ask several development engineers if they know who is responsible for handling the defect. One development engineer responds to this message, by sending a message agreeing to "own" the defect. The development engineer also starts the repair process by pressing another button on the message form to create an initial action item form for the repair task. Later the development engineer starts a related conversation by posting to other engineers a message that raises a design issue concerning the reported defect. From the action item form the engineer may access copies of related defect notifications from the software build system, other mail messages concerning the defect, and the status of the defect repair (as communicated to *Strudel* by the software maintenance tools).

CONCEPTUAL MODEL

The basic types in *Strudel*'s abstract computational model are described in this section, first in overview, then in more detail. *Conversations* and *tasks* are composed of collections of *messages* and *action items*, respectively. A message is used to carry *conversational moves*, for example the move "Request to repair a defect." Conversational moves typically suggest next conversational moves and present *actions* that can be executed by the message's reader. For example the "Request to repair a defect" move suggests "Agree to repair a defect" as a next conversational move. *Conversation types* and *task types* can be defined to guide the way that conversations and tasks are started, developed and ended. For example, a defect resolution conversation defines as initial moves either a "Request to repair a defect" or a question asking "Are you the right person to handle this defect?" A defect tracking task is started either when a *notification* of a defect is reported to *Strudel* by an external tool, such as a product build system, or when a user creates a "Repair Defect" action item. Figure 1 sketches the basic relations between *Strudel's* abstract types, and shows example moves from a defect resolution conversation.

The basic types *conversational-move*, *action-item* and *notification* are subclasses of a root type, namely *task-move*. Instances of these types have a title, optional fields, and actions that the user can apply in the context of the move. Figure 2 shows an example Motif form for a simplified "Repair Defect" action item. Figure 3 shows the type definition for this action item. The type definition in the user's local library supplies default field and action information when the action item is instantiated by the user.

Notifications are treated as pseudo action-items in that they are not instantiated by users but rather in response to a call from an application; otherwise they have the same properties as action items. Figure 4 shows an example notification created by a call from a software build tool to inform the user of a defect.

---

[5]We introduce a conversation type based on the Issue Based Information System (IBIS) methodology [Wern70]. Issues and Positions are sent as specialized messages to others in the design team rather than being added to an argumentation database as in a typical IBIS.

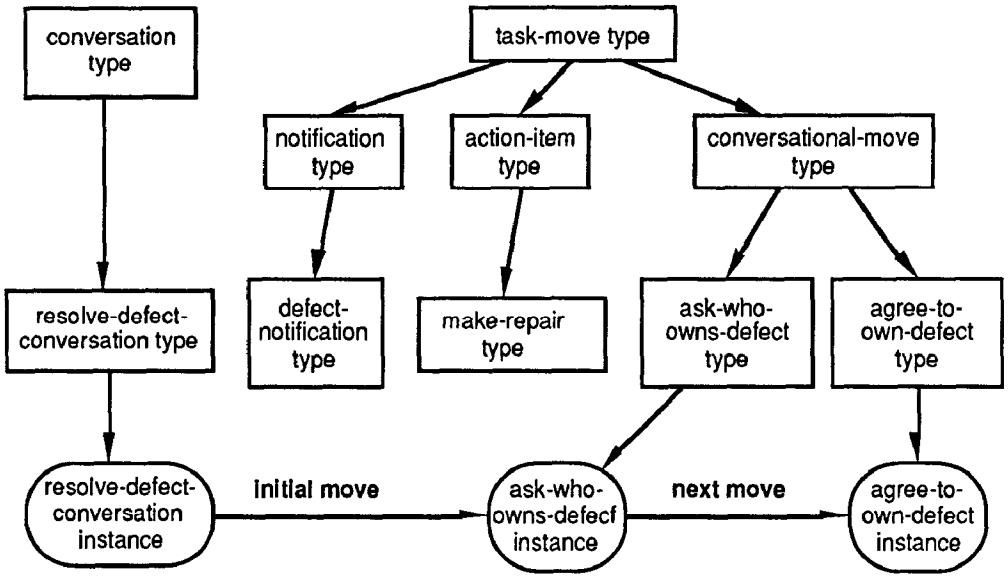

Figure 1. Conversation and move type hierarchy

Actions that are specific to a move type can be defined.[6] These are attached to buttons when the move is presented, for example the "Initiate Repair" button in Figure 2. Actions defined for moves can be used, for example, to draft action items or next conversational moves. Actions defined on fields can be used for example to confirm that a time is free in a calendar or schedule. Actions can also be user-defined to invoke task specific operations on task objects. For example, an action[7] that posts an engineering change order to the design history of a product may be implemented by calling an interface to a production management and inventory control system. An action to retrieve a defective software module may be implemented by a call to a software maintenance tool. Depending on whether a move is being drafted or read, different actions may be presented as specified in the move's type definition. For example the draft message shown in Figure 5 is presented with no actions; whereas the same message when reviewed by the recipient, is presented with the action "Create action item for Defect Repair," as shown in Figure 6. Definitions for actions are given in the recipient's library, or a definition of the action is sent in the message (in this sense messages are self describing).[8]

A *conversational move* may be semi-structured. In this respect, a move is similar to a "semi-structured message" [Malo86]. In Strudel, moves are semi-structured in the sense that each move has a structure composed of fields, but the contents of the fields are not structured. Users can fill in as much or as little information in the fields as they wish, and the information in a field is not necessarily of any specific type. When drafted, conversational moves are inserted into an e-mail message to be sent to other users. A message may contain several moves. A type definition for a conversational move can specify an explicit sequence of suggested next conversational move types, and a preferred or default one. For example, in the message shown in Figure 6, the user is presented with the choices "I am," etc. Figure 7 shows the type definition for this move.

---

[6]For convenience, conversational moves, action items, and notifications are referred to as moves or task moves (as instances of the *task-move* type).

[7]These actions are specified as *Winterp* procedures.

[8]If an action definition is not present locally, *Strudel* will request a copy of the action definition from the message's sender or from the group's server.

Messages are threaded into one or more *conversations*. A conversation consists of an opening and the successor messages to the initial message. The opening specifies the participants, the initial message, and so forth. A user starts a new conversation by selecting a conversational move type to instantiate. This selection is made either from the "Start conversation" top-level menu, or in the context of an existing action item or notification. For example, the initial move "who-will-handle-defect" (shown in Figure 5) is selected from a menu on the "Compose msg" button in the notification shown in Figure 4. The user then edits and sends the message. In responding to a message, users send successor moves thus extending the conversation. A conversation is just this collection of successor messages and the opening descriptor.[9] Users may copy and join the conversation.

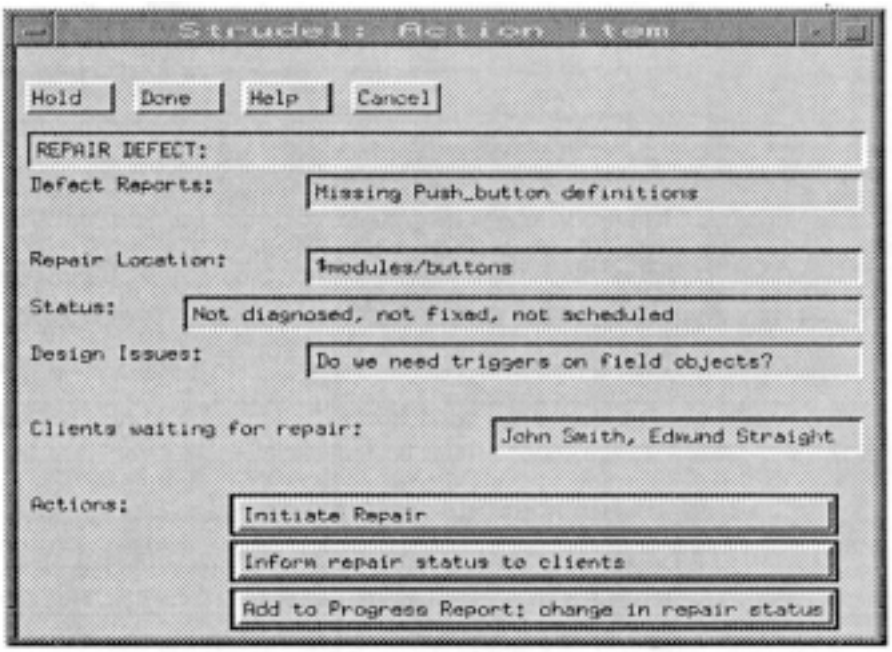

Figure 2. Drafting a "Repair Defect" action item

```
(register-task-move-type 'make-repair
    :title              "Repair Defect"
    :intro              "REPAIR DEFECT:"
    :field-sequence     '(review repair-location status design-issues who)
    :action-types '(    '(start-repair  "Initiate Repair")
                        '(inform-clients "Inform repair status to clients")
                        '(note-in-progress-report
                                "Add to Progress Report: change in repair status")))
```

Figure 3. A type definition for the "Repair Defect" action item

---

[9]In the current prototype, *Strudel* implements the conversation opening by including a conversation id in the message with the initial conversational move. This id is quoted in responding messages.

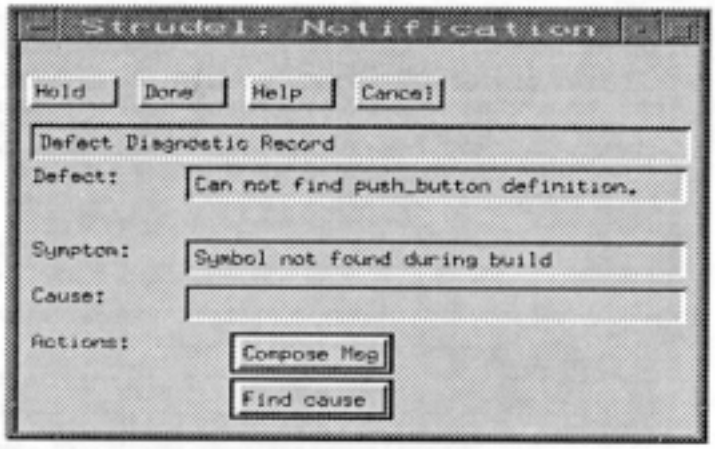

Figure 4. A received notification

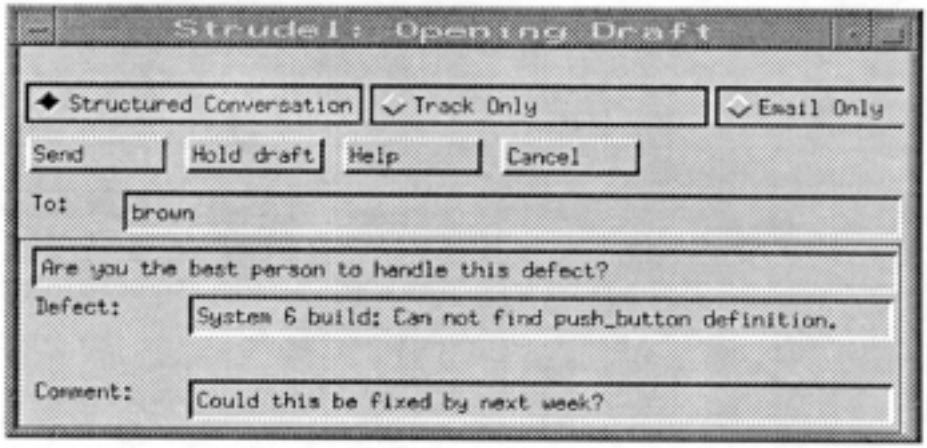

Figure 5. A draft conversational move

Specialized actions for a conversational move allow a user to navigate to the previous and predecessor moves in the conversation,[10] and to linked action items and notifications. Given the ability to navigate, a graphical network of the moves in a conversation can be displayed. Figure 8 shows some of the moves in the defect resolution scenario — the highlighted moves were made; dashed links connect these to moves that could have been made.

A *task* is a collection of action items or notifications that is generated from the initial action item or notification of the task. This is analogous in structure to a *conversation*, that is generated from an initial conversational move. A *conversation* is treated as a specialized task. *Conversation types* and *task types* specify a set of initial move types. Menus of the initial move types declared in conversation and task types are currently used to start new conversations and tasks. Task and conversation types may also specify how next moves are to be drafted, as described in the section "Drafting Messages", below.

---

[10]To allow users to navigate to a previous message which has been lost or deleted, a control move can request a copy of a message to be forwarded, as in Dragon [Come86].

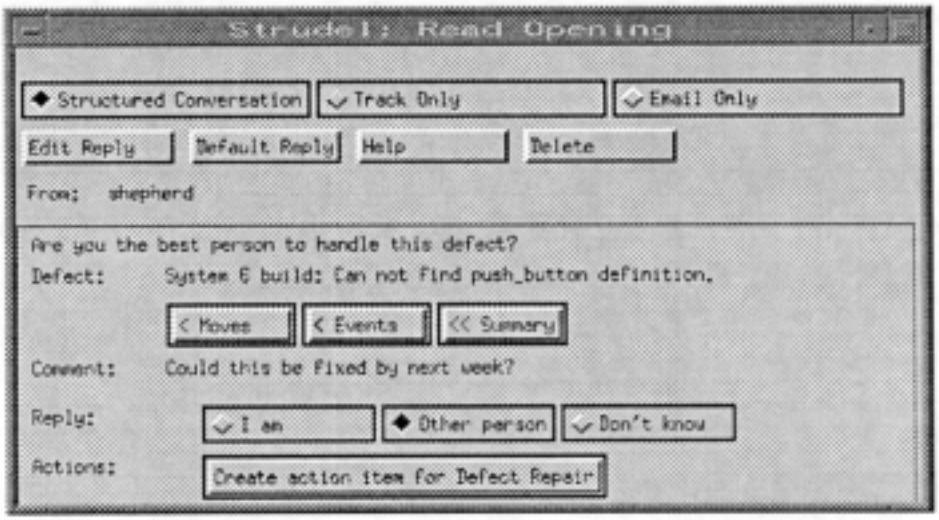

Figure 6. A received conversational move

```
(register-conversational-move-type  'who-will-handle-defect
     :title                                  "Defect owner?"
     :intro                        "Are you the best person to handle this defect?"
     :utterance-sequence                     '(defect  comment)
     :preferred-response-action-type         'other-may-handle
     :next-conv-move-types      '(           '(i-will-handle-defect  "I am")
                                             '(other-may-handle      "Other person")
                                             '(do-not-know            "Don't know"))
  :action-types   '('(make-repair-action     "Create action item for Repair Defect")))

(send who-will-handle-defect :set-field  'defect
     :label                                  "Defect"
     :presentation                           '(string 80 50)
     :read-actions                           '(prev-move  prev-event  summary))
```

Figure 7. A conversational move type definition

Several default presentations of tasks and conversations are provided, including browsers for various classes of action items, notifications, and messages. A simple message browser, which resembles current e-mail browsers, allows conventional e-mail messages and *Strudel* messages to be presented. To present Calendar and "To-Do list" views, browsers will allow users to sort, filter and group action items and messages based on relations for time and keyword matches. For example, messages can be sorted by time sent, grouped by participants, etc. Conversation browsers allow users to navigate within conversations, to archive conversations, etc.

Users can define *overview presentations* for conversations and tasks. Defaults will be indented outlines or simple graphs, as in Figure 8, showing the ordering of moves taken and available. Particularly within a typed conversation or task, a procedure can draft a simplistic specialized *summary* of the conversation or task, for example by assembling from specific fields in several messages a draft of a project's progress report, or the minutes or agenda of a meeting. In the defect resolution scenario, a summary form links related defect reports, a repair status summary, clients waiting for the repair, and so forth.

Drafting Messages

When a user reads a message containing a conversational move, the user can draft a next conversational move, based on the type and content of a predecessor move. The user chooses a next move type from those suggested in the current move or from other library-

defined types, or can send a message containing an untyped move, or can create a new type to use. Conversation types can be extended on the fly by users defining new move types as "next-conv-move-types" in existing move types. In a similar way task types may be defined and evolved.

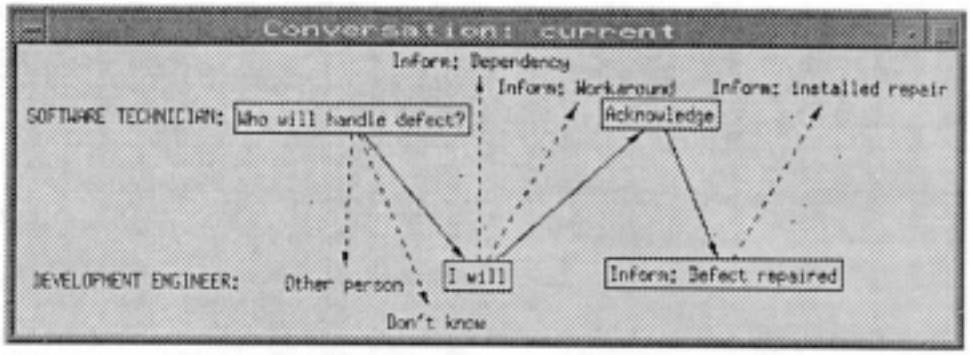

Figure 8. Conversation graph

In the current interface to *Strudel*, the possible next move types are presented via buttons or menus. When reading a received message, the user presses a button to select a next move type. In the defect resolution scenario, Figure 6 shows buttons labelled "I am," "Other person" and "I don't know." A default next move type can be set by the sender of the message, perhaps to indicate a preferred choice which may express a methodology or policy, or just to set a default focus for the discourse. The user chooses either to edit the next move, or to send a default move of the selected type. This next move is "in response to" the current move. This is analogous to the threading created by conventional "in-reply-to" e-mail messages; however the "in response to" move may be addressed to anyone, thus allowing other parties to be entered into the conversation at that point.[11] For example, on receiving a problem description, a user may ask for separate solutions to the problem by sending separate requests to distinct parties.

In order to support drafting of messages using the content of several previous messages, as when summarizing or comparing the content of several previous moves (perhaps from different parties), a message can contain a move which may be "in response to" several moves in several predecessor messages. For example, a "Request to meet" may include agenda items derived from design issues raised in several previous messages. We will need to provide ways for users to at least manually reorder and merge fields in the newly drafted message, in much the same way that users currently cut and paste text when summarizing several previous messages in current interfaces.

Issues

The computational context in which a next conversational move is drafted is dominated by the conversational model. At first glance it appears necessary to support a context containing multiple immediate predecessor moves, in order to thread converging conversations. In order to develop as simple a model as possible, we provided in the current prototype a drafting context that contains only a single immediate predecessor move — the move that the new move is explicitly "in response to." In drafting new

[11] Making a particular move "in response to" a move in a conversation does not prevent the user from making other moves also "in response to" the move. However, the subsequent moves are made "within the sequence of" previously made moves, as in Dragon.

moves, only data in this context are usually accessed, for example in copying the topic of an agenda item or the date of a meeting into a next move. A draft could reflect the state of other moves in the conversation, since other moves can be retrieved given the current move. However, as another means of keeping the model simple, the state of moves in each conversation is currently represented only in each move's state; it is not represented in any global state for the conversation or specific to the conversation type.

In conventional office forms systems, the text labels on fields and buttons are fixed. In *Strudel*, users can easily change the surface text of labels and the default field contents, for example to change the degree of formality in the displayed introductory text. A number of issues arise with this flexibility. The advantage that fixed forms can be understood quickly by the reader is lost. Misunderstandings may be caused by small changes made by a sender since there is the same lack of cues as in conventional e-mail.

An important issue is the development of a framework for interoperability among conversation management systems and coordination systems [Lai88]. Our initial practical approach to this problem has been to design *Strudel* to permit experimental interoperation with conventional "unstructured" e-mail, and other prototype conversation management, active message or coordination systems.[12] We have defined conversational move typing, the addition of fields, and actions orthogonally to each other. This will allow *Strudel* to experimentally interpret messages from other systems that may support one or more of these features. For example, a particular application may send only semi-structured messages, or only unstructured typed messages such as a "Request," or active structured messages. A key issue is providing interoperable references to and descriptions of actions (in messages). For example, if there is a meeting date field in a message, users would like to be able to apply their local definition of an Add-to-Calendar action so that appropriate entries are made in a local calendar, irrespective of the source of the meeting announcement.

CONCLUSION

We have described the conceptual model of the initial *Strudel* toolkit, and its approach to user extensibility of conversation and task objects. We intend for *Strudel* users to gradually evolve groupware extensions to their current work practices by adding semi-structured messages, active messages and conversation management tools as features available in a system that "feels" like a traditional e-mail user agent.

The prototype demonstrates that the basic functionality of *Strudel* can be provided in a small, fast package that is portable, customizable and extensible. With this prototype we will be able to investigate the value to actual user groups of this approach to extensible conversation management. Design and usability issues that we are exploring include the ease of adoption of conversation structuring by work groups, identification of the appropriate "units" of conversation to support, striking the proper balance between user-driven and technology-driven design, integration of computer-based conversations with other collaboration and communication mechanisms in work groups including the use of multi-media, and identification of classes of conversations most appropriate for machine support.

Acknowledgements

Thanks to Susan Brennan, Mark Corscadden, Danielle Fafchamps, Martin Griss, Lars-Erik Hammarin, Nancy Kendzierski, Scott McGregor, Bonnie Nardi, Vicki O'Day, Steve Whittaker and David Williams for helpful discussions.

---

[12]Lee discusses this [Lee89] from the perspective of communication using typed messages.

BIBLIOGRAPHY

[Bore88] Nathaniel S. Borenstein and Chris A. Thyberg. **Cooperative work in the Andrew message system.** In Conf. on Computer-Supported Cooperative Work, pages 306-315,1988.

[Come86] D. Comer and L. Peterson. **Conversation-based mail.** ACM Transactions on Computer Systems, 4(4):299-319, November 1986.

[Deci86] F. De Cindio, G. De Michelis, C. Simone, R. Vassallo, and A. Zaboni. **CHAOS as coordination technology.** In Conf. on Computer-Supported Cooperative Work, pages 325-343, 1986.

[Doll89] Jean Dollimore and Sylvia Wilbur. **Experiences in building a configurable CSCW system.** In Proc. 1st European Conf. on CSCW, pages 215-225, September 1989.

[Fafc90] Danielle Fafchamps, Dave Renolds and Allan Kuchinsky. **The dynamics of small group decision making over the e-mail channel.** To appear in Studies in Computer Supported Cooperative Work: Theory, Practice and Design, ed. J. Bowers and S. Benford. Elsevier.

[Ishii89] Hiroshi Ishii and Kazunari Kubota. **Office procedure knowledge base for organizational office work support.** In B. Pernici and A. A. Verrijn-Stuart, editors, Office Information Systems: The Design Process. Elsevier, 1989.

[Kapl90] Simon Kaplan. **COED: A conversation-oriented tool for coordinated design work.** In Proc. IFIP Int. Workshop on Human Factors in Information Systems, June 1990.

[Wern70] Werner Kunz and Horst Rittel. **Issues as elements of information systems.** Technical Report, Inst. of Urban and Regional Development, Univ. California, Berkeley, July 1970. Working Paper No. 131.

[Krei89] T. Kreifelts, F. Victor, G. Woetzel, and M. Woitass. **A design tool for autonomous group agents.** In Proc. 1st European Conf. on CSCW, pages 204-214, **September 1989.**

[Lee89] Jintae Lee. **How can groups communicate when they use different languages? Translating between partially shared type hierarchies.** Technical Report SSM WP 3076-89-MS, MIT, September 1989.

[Lai88] Kum-Yew Lai and T. Malone. **Object-Lens: A spreadsheet for cooperative work.** In Conf. on Computer-Supported Cooperative Work, pages 115-124, September 1988.

[Maye90] Niels Mayer, Allan Shepherd and Allan Kuchinsky. **Winterp: An object-oriented rapid prototyping, development and delivery environment for building extensible applications with the OSF/Motif UI Toolkit.** In Proc. Xhibition-90 X Window System and Open Systems Technical Conf.erence, pages 49-64, May 1990

[Malo86] T. Malone, K. Grant, K. Lai, R. Rao, and D. Rosenblitt. **Semi-structured messages are surprisingly useful for computer-supported coordination.** In Conf. on Computer-Supported Cooperative Work, pages 102-114, December 1986.

[Moti90] Open Software Foundation. **OSF/Motif Series**, 1990. Prentice-Hall.

[Post82] J. B. Postel. **Standard for the format of ARPA Internet text messages, requests for comments 822.** Technical Report SRI-NIC RFC-822, Stanford Research Institute, August 1982.

[Rose86] M. T. Rose and J. L. Romine. **The Rand MH message handling system: User's manual, UCI Version 6.5 12.** Univ. of California, December 1986.

[Sche88] Robert Scheifler, James Gettys and Ron Newman. **The X Window System: C Library and Protocol Reference.** DEC Press, 1988

[Shep89] Allan Shepherd, Niels Mayer, and Allan Kuchinsky. **Strudel: An electronic conversation toolkit.** Technical Report STL-89-04, Hewlett-Packard Labs, Palo Alto, CA 94303, August 1989.

[Sulo90] Reijo Sulonen and Panu Pietikainen. **Forget-Me-Not — Controlling intercompany operations by intelligent mail.** In Proc. 23rd Hawaii Int. Conf. on Systems Sciences, pages 428–435, 1990.

[Wino87] T. Winograd. **A language/action perspective on the design of cooperative work.** Technical Report STAN-CS-87-1158, or CSLI-87-98, Stanford University, 1987.